MASTER EN SCIENCES INFORMATIQUES

Contribution à la conception d'un réseau privé à commutation par paquets application du recuit simulé

Heinen, Sacha

Award date:
1998

redites !
qu'est-ce qu'une ligne multiparcours ?
* p 51 algorithmes + heuristiques
utilisation d'arbres partagés ?
* p 77 * on ne parle pas de convergence en probabilité

Contribution à la conception d'un réseau privé à commutation par paquets: Application du recuit simulé

Sacha HEINEN

Promoteur:
Monsieur Ph. Van Bastelaer

Mémoire présenté en vue de
l'obtention du titre de
Licencié et Maître en Informatique

Année académique 1997 - 1998

Ici, je veux remercier Mr. Van Bastelaer qui a bien voulu assurer la direction de mon mémoire. C'est son aide constructive qui m'a permis d'aboutir au présent résultat.

Je remercie également Mr. Leclercq qui m'a aidé dans mes recherches des algorithmes.

Pour Christel, Jan et Philip

Table des matières

Chapitre 1

Introduction

Le début de la télécommunication s'est placé sous le signe de la télégraphie et de la téléphonie. Dans le souci d'offrir à tout le monde la possibilité de communiquer avec tout le monde, bon nombre de pays ont confié la communication à distance à un seul organisme public. Toute l'infrastructure et la gestion des télécommunications étaient alors aux mains d'une seule organisation qui détenait le monopole. En Belgique, ce rôle a longtemps été joué par la RTT (aujourd'hui Belgacom).

Depuis quelques dizaines d'années, le domaine des télécommunications a connu des bouleversements technologiques importants, en raison du rapprochement entre la gestion et la communication des données. Sur le plan de la communication, la digitalisation a conduit à la disparition des distinctions entre transmission des données, transmission de la voix et transmission des images. En pratique, cela veut dire que toutes les données peuvent donc être digitalisées. Celles-ci ont alors un même format et peuvent être envoyées sur un même support. Comme les organisations publiques de téléphonie possédaient les infrastructures nécessaires à la télécommunication, ce sont elles qui ont contrôlé le marché des télécommunications. Aujourd'hui, ce privilège est en voie de disparition à cause de l'apparition de nouvelles entreprises.

La transmission des données à travers un réseau de communication et la forme des messages transmis peuvent se présenter de différentes façons. Des technologies de plus en plus sophistiquées et des infrastructures de plus en plus fiables ont, en effet, permis d'augmenter progressivement les performances générales d'un réseau. Parmi ces performances, on peut citer : la diminution des temps de transmission, l'augmentation de la capacité de transmission, la diminution du taux d'erreurs et la diminution des disparitions d'informations.

1.1. Définition du problème

Le domaine des télécommunications évolue tellement vite que la notion des performances doit être envisagée tant au niveau physique qu'au niveau conceptuel. Chacune de ces branches de la téléinformatique mérite une étude approfondie dépassant largement le cadre du présent travail. Néanmoins, nous tenterons de clarifier un point qui nous semble intéressant dans le monde des télécommunications : la conception de réseaux.

Avant d'installer un réseau, on veille à étudier tous les éléments y intervenant. Cela va de la détermination des composantes physiques au choix des protocoles et de l'endroit où il faut placer les différentes composantes. C'est précisément au placement des composantes que nous nous limiterons. Le nombre de réseaux de types différents étant assez important, nous nous focaliserons, en outre, sur un type bien particulier de réseaux : les réseaux à commutation par paquets pour une entreprise privée. La topologie du réseau considéré dans le cadre de ce mémoire se repartira sur toute la Belgique.

Contrairement aux réseaux à commutation de circuits, ces réseaux n'établissent pas de connexion directe entre deux hôtes. Il n'est donc pas nécessaire de réserver la totalité de la capacité de transmission des lignes le long d'une liaison à travers le réseau. Au contraire, les données sont envoyées dans une séquence de paquets (*Figure 1.1*). Chaque paquet est passé par le réseau de noeud à noeud, le long de la liaison de l'hôte source à l'hôte destination.

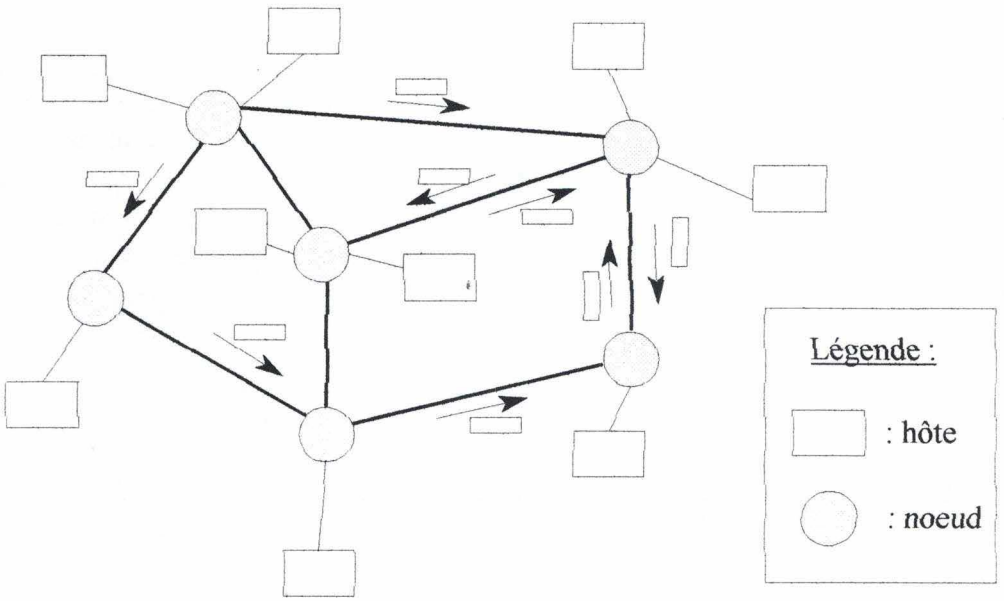

Figure 1.1 : Réseau à commutation par paquets

La liaison entre deux hôtes n'est donc utilisée qu'au moment du transfert. Par conséquence, on ne paie que le temps d'utilisation, c'est-à-dire le temps de la communication réelle, et non la capacité totale réservée au moment de la location de la ligne. Même si les avantages des réseaux à commutation par paquets (flexibilité, distribution des ressources et stabilité) entraînent des coûts supplémentaires, ces réseaux s'imposent lorsque le taux d'occupation est relativement faible.

Une des techniques permettant de faire passer plusieurs communications (ici, les paquets) sur une même ligne est le multiplexage. Elle consiste à regrouper les paquets de plusieurs lignes pour les envoyer sur une seule ligne (*Figure 1.2*). Cette ligne aboutit à un démultiplexeur effectuant l'opération dans le sens inverse : il répartit le signal arrivant sur les lignes correspondantes.

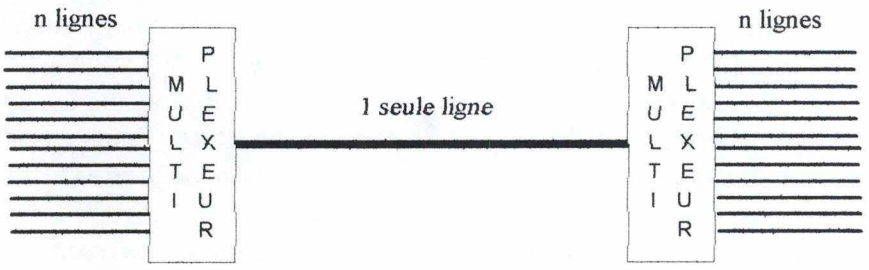

Figure 1.2 : Principe du multiplexage

On distingue deux types de multiplexage : le multiplexage par fréquence et le multiplexage temporel.

La méthode du multiplexage par fréquence permet de faire passer plusieurs signaux sur une ligne lorsque le signal transmis sur le réseau est un signal analogique (par exemple le RTC - réseau téléphonique commuté).

En revanche, si le signal envoyé sur le réseau est numérique, on applique le multiplexage temporel pour transmettre plusieurs signaux. C'est cette technique qui nous intéresse ici. Les signaux digitaux transmis sur une seule ligne de communication par petits morceaux sont envoyés à tour de rôle. Dans un multiplexeur temporel, les données venant de leur source sont stockées pendant un court instant dans un tampon. Ces tampons sont lus séquentiellement pour former une chaîne composée de données digitales (*Figure 1.3a*). La capacité de la ligne de sortie doit donc au moins être égale à la somme des vitesses de transmission des signaux d'entrée. Cette technique est appelée le multiplexage statique.

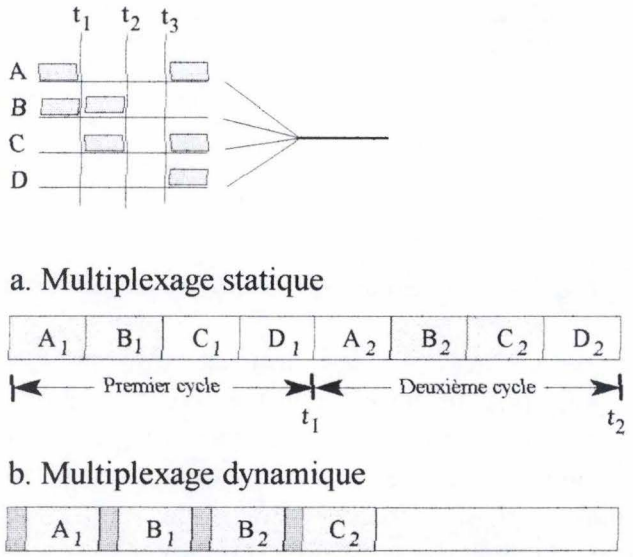

Figure 1.3 : Multiplexage statique et multiplexage dynamique

Elle présente un inconvénient majeur : à chaque cycle de lecture dans les tampons, la partie correspondante sur la ligne de sortie est réservée. Il y a donc des laps de temps sur la ligne de sortie qui ne contiennent pas de données. Ainsi, le multiplexage statique utilise toute la ligne sans exploiter complètement la capacité de la ligne. L'autre possibilité est le multiplexage dynamique. Il effectue une allocation dynamique des laps de temps de la ligne de sortie, en fonction de la demande des tampons des lignes d'entrée. A l'entrée, le multiplexeur lit les données des tampons pour construire une chaîne et envoyer cette chaîne sur la ligne de sortie (*Figure 1.3b*).

De l'autre coté de la ligne multiplexée, les données arrivent de manière imprévisible dans le démultiplexeur. Pour assurer une répartition adéquate des paquets entrants sur les différentes lignes de sortie, il faut "adresser" ces communications, c'est-

à-dire que chaque paquet doit contenir une adresse. Les réseaux à commutation par paquets utilisent le multiplexage temporel dynamique.

Il est à noter que la transmission de paquets sur les lignes du réseau à commutation par paquets se fait en full-duplex. Lors d'une liaison full-duplex, les deux stations peuvent simultanément envoyer et recevoir des données.

Les réseaux à commutation par paquets sont constitués de noeuds reliés entre eux par des lignes de transmission. Les paquets qui transitent sur chaque ligne peuvent avoir des destinations différentes. Chaque noeud doit donc router tout paquet entrant vers la ligne de sortie, qui l'orientera vers sa destination.

Les techniques de routage définissent les chemins que doivent suivre les différents paquets à partir de caractéristiques générales du réseau telles que la topologie et le trafic sur les lignes de transmission. Le routage se traduit, au niveau de chaque noeud, par une table de routage, dont la consultation permettra au noeud de déterminer sur quelle ligne de sortie il doit émettre un paquet pour que celui-ci atteigne sa destination. L'ensemble de toutes les tables de routage forme la matrice de routage où chaque ligne représente la table de routage d'un noeud précis.

Lors de la phase de conception, il est préférable d'adopter la technique du routage fixe, même si lors de l'utilisation du réseau, on optera pour le routage adaptatif. Les règles d'acheminement du routage fixe sont, en principe, établies une fois pour toutes et visent à optimiser le critère de performance. Quant au routage adaptatif, il consiste à calculer dynamiquement l'acheminement des paquets en fonction du taux d'occupation des lignes. Il ne sera pas développé dans le présent mémoire puisqu'il est difficile de tenir compte de l'évolution du trafic durant l'utilisation du réseau pendant la phase de conception; seuls des modèles mathématiques complexes le permettent.

L'algorithme de conception utilisé dans ce mémoire déterminera l'endroit exact où on placera les noeuds, la position et la capacité des lignes hôte-noeud et des lignes noeud-noeud ainsi que la matrice de routage en tentant d'optimiser le coût global du réseau et en garantissant en même temps des performances raisonnables. Pour trouver l'endroit où placer les noeuds, l'algorithme peut choisir un sous-ensemble de noeuds parmi un ensemble de positions potentielles proposées par le concepteur du réseau. Connaissant la position des lignes et la matrice de routage, on peut calculer la capacité "théorique" de chacune de ces lignes. L'entreprise peut alors louer des lignes "physiques" d'une capacité réalisable supérieure à la capacité théorique présente chez le fournisseur. Le coût de location de ces lignes déterminera alors le coût total du réseau (la somme des coûts de toutes les lignes du réseau). Nous calculerons également les temps de réponse sur chaque ligne.

1.2. Aperçu du travail

Le chapitre suivant cette introduction sera consacré aux méthodes de calcul des capacités et des performances d'un réseau. Cela comprend la description et le calcul du trafic ainsi que le calcul du coût et le calcul des performances de ce réseau. A partir de la matrice de communication, il est possible - en supposant que la conception de réseaux ait trouvé la position des noeuds et des lignes - de définir le trafic moyen par ligne. Connaissant le taux d'occupation et la taille des paquets, on peut en déduire la matrice des capacités de lignes. Partant du tarif des lignes et de la longueur des lignes, on trouve alors le coût de chacune de ces lignes. Enfin, on exprimera les performances du réseau par le temps de transmission sur les lignes et sur les liaisons entre hôtes. Tous ces calculs seront illustrés par un exemple.

Au chapitre trois, nous survolerons quelques algorithmes présents dans la littérature et pouvant être appliqués à la conception d'un réseau à commutation par paquets. Ces algorithmes se donnent comme objectif de réduire les coûts le plus possible tout en garantissant des performances raisonnables. Ils seront expliqués brièvement pour donner une idée générale de leur fonctionnement. Il s'agit de l'algorithme d'insertion, de l'algorithme d'élimination, de l'algorithme de placement des noeuds, de l'algorithme des réseaux d'accès, de l'algorithme de Steiglitz, de l'algorithme dual de l'arbre minimal, de l'algorithme de la découpe en deux sous-réseaux (Cutset Algorithm) et de l'algorithme du recuit simulé (Simulated Annealing Algorithm). Afin de faciliter la compréhension, la plupart des algorithmes s'accompagnent d'un exemple.

Parmi ceux-ci, c'est l'algorithme du recuit simulé que nous avons retenu pour trouver la topologie des réseaux à concevoir. Le chapitre quatre expliquera donc le recuit simulé plus en détail - d'abord son origine (la physique thermodynamique), ensuite sa modélisation en général et enfin son application aux réseaux.

Le cinquième chapitre donnera les éléments d'implémentation de l'algorithme du recuit simulé, illustrera par un exemple les résultats de l'application de l'algorithme et discutera les résultats obtenus.

Ce mémoire se termine par la conclusion.

Chapitre 2

Méthodes de calcul des capacités et des performances d'un réseau

La conception de réseaux de télécommunication a pour objectif de déterminer la topologie d'un réseau en fonction de certains critères en respectant certaines contraintes imposées au concepteur. Le problème principal consiste à minimiser le coût total du système tout en garantissant des performances raisonnables en ce qui concerne le temps de réponse, la sécurité et la disponibilité du réseau. Le coût fait l'objet des méthodes analytiques, tandis que les aspects de qualité et de performance sont principalement étudiés dans le cadre des simulations de réseaux.

Ce chapitre expliquera les concepts sous-jacents aux méthodes analytiques de la conception de réseaux et abordera les notions de performance.

2.1. Trafic d'un réseau

Dans une entreprise, l'information doit circuler. Ce besoin de communication existe non seulement entre individus, mais aussi entre ordinateurs - donc au niveau informatique et au niveau réseau.

Les besoins de communication d'une entreprise détermineront la structure d'un tel réseau. Seule une analyse détaillée de ces besoins permet une conception précise du réseau. Le but d'une telle analyse est de spécifier le trafic qu'il y aura sur le réseau.

Réseau de télécommunication

Une décomposition du réseau de télécommunication en composantes permet une meilleure perception du problème de la conception d'un réseau. Les composantes de base sont les suivantes :

Hôtes : Les hôtes sont des lieux entre lesquels il y a des besoins de communication. Chaque hôte est défini par ses coordonnées et par ses besoins de communication avec les autres hôtes.

Liaisons : Une liaison entre un hôte A et un hôte B est une communication permettant à l'hôte A d'émettre des paquets vers l'hôte B. Généralement, à une liaison A→B correspond une liaison B→A. Ces liaisons représentent la communication sur le plan logique; une liaison est donc une entité logique entre deux hôtes.
Toute liaison est caractérisée par les besoins de communication - p. ex. par un trafic moyen - entre deux hôtes.

Lignes : Les lignes d'un réseau de communication représentent le moyen de transport par lequel les données sont transportées entre les différents noeuds et/ou hôtes. Les lignes décrivent le réseau de communication au

niveau physique. Elles sont caractérisées par leur capacité, leur longueur et leur coût. Une liaison emprunte un certain nombre de lignes.

Noeuds : Les noeuds n'envoient pas de trafic sur le réseau. Ce sont seulement des commutateurs (routeurs) qui effectuent le routage dans le réseau de communication. Ils sont définis par leurs coordonnées et les lignes vers lesquelles ils peuvent router les paquets.

Machines : Une machine est un noeud ou un hôte. Parfois, une machine peut à la fois être un noeud et un hôte, c'est-à-dire émettre ses propres paquets et router les paquets des autres hôtes. Dans ce cas, on précisera à chaque fois la fonctionnalité de la machine.

La *Figure 2.1* montre un exemple d'un réseau de télécommunication exprimé dans ses composantes.

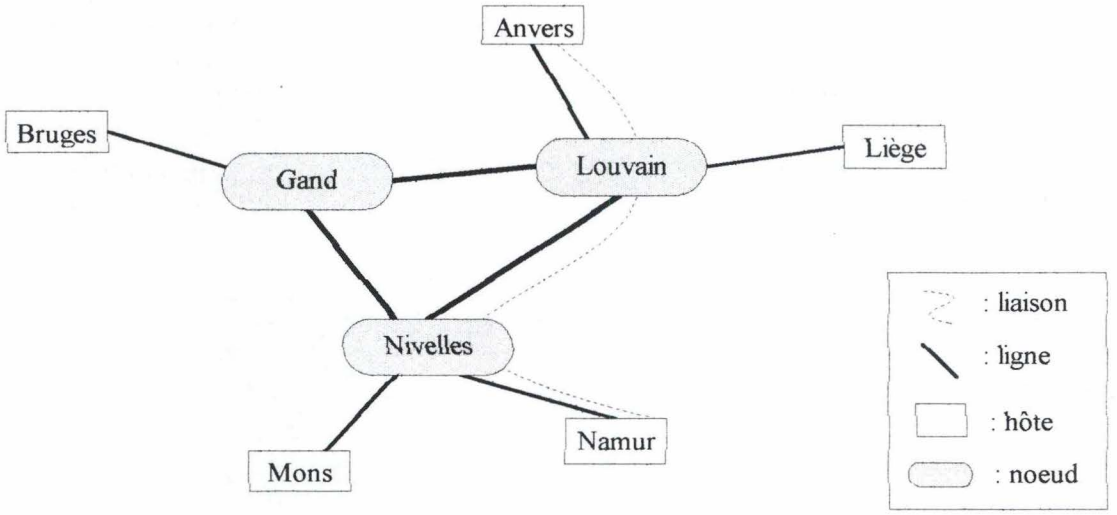

Figure 2.1 : Exemple d'un réseau de communication

Le trafic sur une liaison prend généralement une forme aléatoire et variable en fonction du temps (*Figure 2.2*). On ne peut donc pas prédire le trafic exact entre hôtes à tout instant. C'est pourquoi les besoins de communication sont généralement estimés en moyenne, plus précisément par le trafic moyen entre hôtes. S'il n'est pas possible de prédire exactement le trafic moyen, il est plus simple d'en avoir une prévision raisonnable.

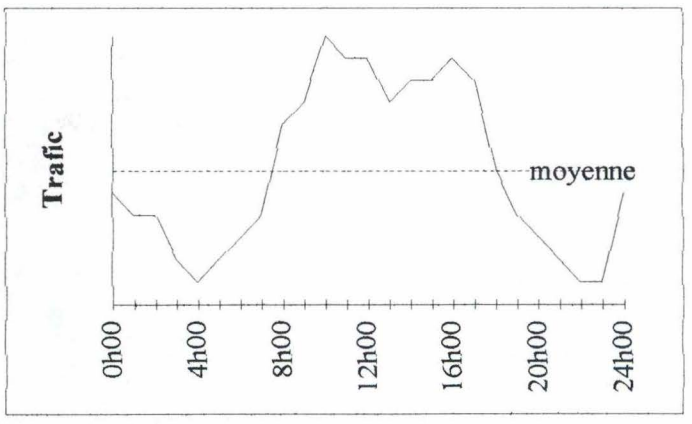

Figure 2.2 : Distribution du trafic

Le trafic de tout le réseau est généralement résumé dans la matrice de communication. Cette matrice constitue l'origine de toutes les étapes de conception et d'optimisation d'un réseau. L'établissement de la matrice de communication est donc une étape importante de la conception.

Matrice de communication

La matrice de communication décrit les flux de communication moyens (le trafic) d'une manière succincte, claire et précise.

Plus précisément, elle exprime les flux de communication sur toutes les liaisons. Les hôtes "origine" déterminent les indices de ligne et les hôtes "destination" les indices de colonne. Chaque élément M_{ij} de la matrice de communication M représente le trafic moyen de l'hôte d'origine i vers l'hôte de destination j (*Tableau 2.1*).

Il est évident que la diagonale de la matrice de communication ne comprend que des valeurs nulles.

Matrice de communication (en paquets/sec.)					
Hôte dest. / Hôte orig.	Bruges	Anvers	Liège	Namur	Mons
Bruges	0	60	30	20	20
Anvers	50	0	50	30	20
Liège	40	40	0	50	30
Namur	20	20	60	0	50
Mons	30	20	40	60	0

Tableau 2.1 : Matrice de communication

Dans le cas d'un réseau à commutation par paquets, ce trafic est réalisé par l'envoi de paquets dont, en principe, tant la longueur que l'intensité varient de façon aléatoire. On supposera dans le cadre de ce mémoire la longueur fixe (p. ex. 250 bytes) et on se basera sur le nombre moyen de paquets émis par seconde.

Si l'on fait la somme de toutes les composantes de la matrice de communication, on obtient le nombre moyen de paquets circulant par seconde sur l'ensemble du réseau, ce qui caractérise l'intensité de son emploi. Dans l'exemple de la *Figure 2.1*, le trafic total est de 740 paquets par seconde.

Matrice de routage

Le chemin que chaque paquet prend sur une liaison entre deux hôtes et, par conséquent, les noeuds qu'il traverse sur le chemin, dépend du type de routage choisi. Lorsque le routage est adaptatif, la suite des noeuds par lesquels les paquets passent sur une liaison, change en fonction de la disponibilité des lignes et des noeuds du réseau. Dans le cas du routage fixe, la décision de routage est précisée de façon plus ou moins permanente.

Dans le cadre de ce mémoire, on utilise le routage fixe. Le chemin que les paquets prennent sur une liaison est donc unique et connu dès que le paquet quitte l'hôte "origine". Les noeuds doivent savoir vers quelle machine envoyer le paquet pour arriver à une destination. Cette information se trouve dans les tables de routage des noeuds. Dans le cas du routage fixe, elles sont initialisées dès le départ et leur contenu est stable.

Pour faciliter les calculs qui suivent, nous allons regrouper les tables de routage de chaque noeud dans une matrice de routage. Cette matrice permet une meilleure perception des décisions de routage. Dans le cas du réseau de la *Figure 2.1*, la table de routage à Gand, par exemple, indique la machine vers laquelle le noeud doit envoyer les paquets pour atteindre la destination (*Tableau 2.2*).

Table de routage	
Destination	Machine
Bruges	Bruges
Anvers	Louvain
Liège	Louvain
Namur	Nivelles
Mons	Nivelles

Tableau 2.2: Table de routage à Gand

Les tables de routage de Louvain et Nivelles peuvent être trouvées de façon semblable. En plus, connaissant la topologie, on établit un tableau indiquant à quel noeud chaque hôte est connecté (Tableau 2.3).

Tableau des lignes hôte-noeud					
Hôte / Noeud	Bruges	Anvers	Liège	Namur	Mons
Gand	×				
Louvain		×	×		
Nivelles				×	×

Tableau 2.3 : Tableau indiquant la connexion entre hôte et noeuds

En comparant toutes les tables de routage avec le tableau des lignes hôte-noeud, on trouve la matrice de routage du réseau de la *Figure 2.1* :

Matrice de routage			
Noeud / Noeud	Louvain	Nivelles	Gand
Louvain	/	d	d
Nivelles	d	/	d
Gand	d	d	/

Tableau 2.4 : Matrice de routage

L'élément R_{ij} de la matrice de routage R (*Tableau 2.4*) désigne le noeud vers lequel le noeud i doit envoyer les paquets pour joindre le noeud j. Le *d* signifie que les paquets seront envoyés directement vers la destination sans passer par un noeud intermédiaire. Les éléments de la diagonale ne contiennent évidemment pas de valeur. Etant donnée la taille du réseau, la matrice de routage n'est ici composée que de connexions directes.

Matrice du trafic moyen par ligne

Le trafic moyen sur une liaison produit un trafic de même valeur sur chacune des lignes composant cette liaison. Pour chaque ligne, le trafic moyen qui la traverse est donc la somme des trafics moyens de toutes les liaisons qui empruntent cette ligne.

Ce trafic moyen se déduit de la matrice de communication et des décisions de routage. La règle de routage qu'on applique dans le cadre de ce mémoire est la suivante: chaque noeud envoie les paquets sur les lignes pour lesquelles la distance à la destination est minimale.

Le *Tableau 2.5* indique le chemin que les paquets d'une liaison prennent en traversant le réseau. Les hôtes "origine" déterminent les indices de ligne et les hôtes "destination" les indices de colonne. Ainsi, pour le réseau de la *Figure 2.1*, les paquets venant de Bruges et destinés à Namur vont d'abord de Bruges à Gand. Comme Namur est connecté au noeud de Nivelles, les paquets doivent traverser le réseau de Gand à Nivelles. Le noeud à Gand doit alors prendre une décision de routage. Sa table de routage (*Tableau 2.2*) indique qu'il y a une connexion directe avec Nivelles. Les paquets entre Bruges et Namur empruntent donc les lignes Bruges-Gand, Gand-Nivelles et Nivelles-Namur.

Matrice des lignes par liaison					
Hôte / Hôte	Bruges	Anvers	Liège	Namur	Mons
Bruges	/	Br-Gd, Gd-Lv, Lv-Av	Br-Gd, Gd-Lv, Lv-Lg	Br-Gd, Gd-Nv, Nv-Na	Br-Gd, Gd-Nv, Nv-Mo
Anvers	Av-Lv, Lv-Gd, Gd-Br	/	Av-Lv, Lv-Lg	Av-Lv, Lv-Nv, Nv-Na	Av-Lv, Lv-Nv, Nv-Mo
Liège	Lg-Lv, Lv-Gd, Gd-Br	Lg-Lv, Lv-Av	/	Lg-Lv, Lv-Nv, Nv-Na	Lg-Lv, Lv-Nv, Nv-Mo
Namur	Na-Nv, Nv-Gd, Gd-Br	Na-Nv, Nv-Lv, Lv-Av	Na-Nv, Nv-Lv, Lv-Lg	/	Na-Nv, Nv-Mo
Mons	Mo-Nv, Nv-Gd, Gd-Br	Mo-Nv, Nv-Lv, Lv-Av	Mo-Nv, Nv-Lv, Lv-Lg	Mo-Nv, Nv-Na	/

Tableau 2.5 : Matrice des lignes par liaison

Le trafic moyen ($T_{Gd,Nv}$) sur la ligne Gand-Nivelles (direction Gand→Nivelles) est donc la somme des trafics moyens des liaisons Bruges-Mons ($M_{Br,Mo}$) et Bruges-Namur ($M_{Br,Na}$) (*Tableau 2.5*) :

$$T_{Gd\text{-}Nv} = M_{Br,Mo} + M_{Br,Na} = 20 \text{ paq./sec.} + 20 \text{ paq./sec.} = 40 \text{ paq./sec.}$$

Le trafic moyen par ligne est souvent résumé dans la matrice du trafic moyen par ligne (*Tableau 2.6*). Chaque élément T_{ij} de cette matrice T représente le trafic moyen d'une machine i vers une machine j. Le trafic est nul pour les machines qui ne sont pas reliées directement.

Matrice du trafic moyen par ligne (en paquets/sec.)								
Machine / Machine	Bruges	Anvers	Liège	Namur	Mons	Gand	Louvain	Nivelles
Bruges	0	0	0	0	0	130	0	0
Anvers	0	0	0	0	0	0	150	0
Liège	0	0	0	0	0	0	160	0
Namur	0	0	0	0	0	0	0	150
Mons	0	0	0	0	0	0	0	150
Gand	140	0	0	0	0	0	90	40
Louvain	0	140	180	0	0	90	0	130
Nivelles	0	0	0	160	120	50	140	0

Tableau 2.6 : Matrice du trafic par ligne

Ce calcul du trafic moyen sur chaque ligne va nous permettre de calculer les capacités des lignes à installer.

Matrice des connexions :

La matrice des connexions exprime pour chaque paire de machines la capacité de la ligne du réseau qui les relie directement (*Tableau 2.7*). Par capacité, on entend ici la capacité que la ligne doit avoir pour pouvoir véhiculer le trafic moyen (trouvé ci-dessus). Par conséquent, cette capacité est nulle pour les machines qui ne sont pas reliées directement. Le calcul des capacités est expliqué ci-dessous.

Dans le cadre de ce mémoire, on a pris l'hypothèse que les lignes sont full-duplex. Par conséquent, on peut dire que cette matrice est symétrique, c'est-à-dire que la capacité de la ligne reliant la machine i à la machine j est la même dans les deux sens de transmission. Elle se base sur le maximum entre le trafic moyen de la machine i vers la machine j et le trafic moyen de j vers i.

Matrice des connexions (en bps)								
Machine / Machine	Bruges	Anvers	Liège	Namur	Mons	Gand	Louvain	Nivelles
Bruges	0	0	0	0	0	466667	0	0
Anvers	0	0	0	0	0	0	500000	0
Liège	0	0	0	0	0	0	600000	0
Namur	0	0	0	0	0	0	0	533333
Mons	0	0	0	0	0	0	0	500000
Gand	466667	0	0	0	0	0	300000	166667
Louvain	0	500000	600000	0	0	300000	0	466667
Nivelles	0	0	0	533333	500000	166667	466667	0

Tableau 2.7 : Matrice des connexions

La capacité de chaque ligne (exprimée en bps -"bits par seconde") est calculée en fonction du trafic moyen sur cette ligne et d'un taux d'occupation désiré (ρ) que nous fixerons par exemple ici à la valeur de 60%. Remarquons que l'étude des files d'attente a montré qu'un taux d'occupation d'au plus 60% garantit le bon déroulement des transmissions de messages et assure que le réseau ne sera pas surchargé. La formule (2.1.) exprime cette relation.

$$C = \frac{\lambda \cdot l}{\rho} \qquad (2.1.)$$

où C est la capacité (en bps)
λ est le trafic moyen (en paquets/sec.)
l est la longueur des paquets (en bits/paq.)
ρ est le taux d'occupation désiré : 0,6

Comme déjà mentionné plus haut, les paquets sont supposés de longueur fixe pour simplifier les calculs. La longueur des paquets l est donc une constante (ici 250 bytes, càd 2000 bits).

La capacité de la ligne Bruges-Gand, par exemple, s'exprime de la manière suivante :

$$C_{Br,Gd} = \max (C_{Br \to Gd}, C_{Gd \to Br})$$
$$= \max \left(\frac{130 \ {}^{paq.}/_{sec.} \cdot 2000 \ {}^{bits}/_{paq.}}{0,60}, \frac{140 \ {}^{paq.}/_{sec.} \cdot 2000 \ {}^{bits}/_{paq.}}{0,60} \right)$$
$$= \frac{280.000 \text{ bps}}{0,60} = 466.667 \text{ bps}$$

Le *Tableau 2.7* montre la matrice des connexions N dont chaque élément N_{ij} représente les capacités de ligne entre une machine i et une machine j.

La capacité des lignes physiques à installer entre les hôtes et noeuds se calcule en arrondissant la capacité "théorique" de la ligne correspondante à la capacité réalisable supérieure vu le catalogue du fournisseur des lignes. Nous allons dans ce mémoire utiliser les lignes offertes par Belgacom, actuellement un fournisseur parmi plusieurs en Belgique. Ce fournisseur offre des lignes par multiples de 64 Kbps allant de 64 Kbps à 2 Mbps.

Matrice des capacités réalisables (en Kbps)								
Machine / Machine	Bruges	Anvers	Liège	Namur	Mons	Gand	Louvain	Nivelles
Bruges	0	0	0	0	0	512	0	0
Anvers	0	0	0	0	0	0	512	0
Liège	0	0	0	0	0	0	640	0
Namur	0	0	0	0	0	0	0	576
Mons	0	0	0	0	0	0	0	512
Gand	512	0	0	0	0	0	320	192
Louvain	0	512	640	0	0	320	0	512
Nivelles	0	0	0	576	512	192	512	0

Tableau 2.8 : Matrice des capacités réalisables

Le *Tableau 2.8* montre la matrice des capacités réalisables L déduite de la matrice des connexions du *Tableau 2.7*. Chaque élément L_{ij} représente la capacité réalisable de la ligne entre la machine i et la machine j. Si les machines ne sont pas reliées directement, cette capacité est nulle.

Connaissant la capacité réalisable de chaque ligne, on peut maintenant calculer le coût de chaque ligne et par conséquent le coût global du réseau de télécommunication.

2.2. Coûts d'un réseau

La minimisation du coût d'un réseau - tout en respectant des performances raisonnables - est le point central de la conception d'un réseau de télécommunication. Ce coût peut être décomposé en trois parties (*Figure 2.3*) :

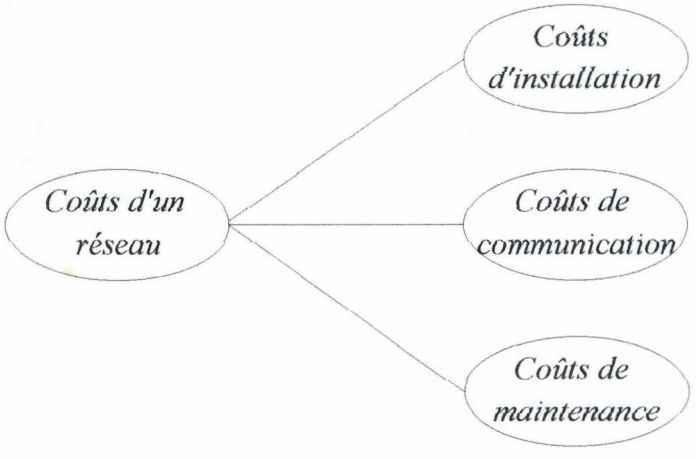

Figure 2.3 : Structure des coûts d'un réseau

Dans le cas d'un réseau privé avec lignes louées, ces trois parties se présentent sous la forme suivante :

Coûts d'installation : Les coûts d'installation comprennent tous les frais d'étude, les coûts d'installation de l'infrastructure du réseau ainsi que la configuration de cet équipement.

Coûts de maintenance : Les coûts de maintenance représentent le montant nécessaire à l'entretien et la gestion du réseau et des noeuds.

Coûts de communication : Ces coûts de communication désignent les coûts d'utilisation des services offerts par un fournisseur de lignes de communication. Ils dépendent de la fréquence de transmission de données sur le réseau et de la répartition de ces données dans le temps.

Remarquons que les frais d'installation sont uniques et que les coûts de communication et de maintenance sont des coûts mensuels.

Coûts d'installation

Une entreprise qui désire mettre en place un réseau privé doit étudier ses propres besoins de communication. Après cette analyse, l'entreprise peut elle-même effectuer la conception du réseau. Les frais de ces études constituent une première composante des coûts d'installation.

Après cette phase de conception, l'entreprise doit installer l'équipement nécessaire pour que le réseau fonctionne. La mise en place de cet équipement comprend l'achat des noeuds (routeurs), des machines hôtes,... et leur configuration. Cette phase prend une part importante dans les coûts d'installation.

En ce qui concerne les lignes, on fait généralement appel à un fournisseur de services de réseau qui possède une infrastructure répondant aux exigences de l'entreprise. Si l'entreprise n'est pas raccordée aux lignes du fournisseur, elle doit, en plus, mettre en place des lignes à partir de son bâtiment vers le point d'accès des lignes. Les frais de cette installation font aussi partie des coûts d'installation.

Tous ces frais sont des frais fixes qui s'imposent avant l'utilisation du réseau.

Coûts de maintenance

Comme tout système informatique, le réseau doit être géré et entretenu. La gestion du réseau est le facteur principal des coûts de maintenance. Elle couvre, entre autres, la gestion des utilisateurs (lorsqu'ils changent de département par exemple) et

des applications "réseau". Comme cette gestion doit être assurée de manière continue, ces coûts sont calculés sur base mensuelle.

A ces coûts de maintenance s'ajoutent l'entretien des noeuds (le hardware) et la gestion des noeuds du réseau (changements de configuration). Ces frais sont également mensuels et doivent être facturés pour chaque noeud du réseau.

Coûts de communication

Lors de la conception d'un réseau de communication, l'estimation des coûts de communication est d'une importance majeure, puisqu'ils créent la plus grande partie des frais mensuels. En plus, ces coûts dépendent fortement de la structure du réseau, puisqu'ils dépendent de la longueur et de la capacité des lignes. En optimisant la structure du réseau, on va également diminuer les coûts de communication du réseau. Dans le programme accompagnant ce mémoire, on vise donc à minimiser ces coûts en focalisant les efforts sur la recherche d'une topologie de réseau optimale.

Une entreprise qui désire mettre en place un réseau privé fait généralement appel à un fournisseur de services de réseaux pour emprunter des lignes. L'utilisation de cette infrastructure de la part de l'entreprise est en général facturée sur base mensuelle par le fournisseur.

Dans le cas d'un abonnement à des lignes (c'est-à-dire des lignes louées), toute la capacité d'une ligne du réseau est réservée à l'utilisateur pour une durée indéterminée. Le coût d'abonnement à une ligne dépend de la capacité de cette ligne. Comme l'utilisation de toute la ligne est facturée, cette solution n'est intéressante que lorsque le trafic sur le réseau est élevé sur une longue période.

Le coût de transmission de données sur une seule ligne est une fonction de la capacité et de la longueur de la ligne, c'est-à-dire la distance entre deux machines adjacentes :

$$C = f(\text{capacité},\text{distance}) \qquad (2.2.)$$

Dans ce travail, l'intérêt porte particulièrement sur les réseaux en Belgique. Le coût des lignes est donc déterminé par les tarifs du fournisseur de services de réseau. Comme nous allons utiliser les lignes du fournisseur Belgacom, on doit se baser sur leurs tarifs. Actuellement, ce système de tarification fait la distinction entre tarif zonal et tarif interzonal. Par conséquent, il y a des lignes de type zonal et de type interzonal.

La *Figure 2.4* montre un exemple d'une ligne entre deux sites situés dans deux zones distinctes. Les machines sont d'abord reliées au BCT (centre local) le plus proche.

L'accès à ce centre est facturé par le montant (montant fixe) d'accès aux services proposés par le fournisseur. Comme chaque BCT est connecté au centre zonal le plus proche, la facturation dépend des distances à ces centres zonaux (d_1 et d_2). Finalement, les centres zonaux sont reliés entre eux par une distance d_i. Le coût d'une ligne entre deux machines sera donc lié à ces distances.

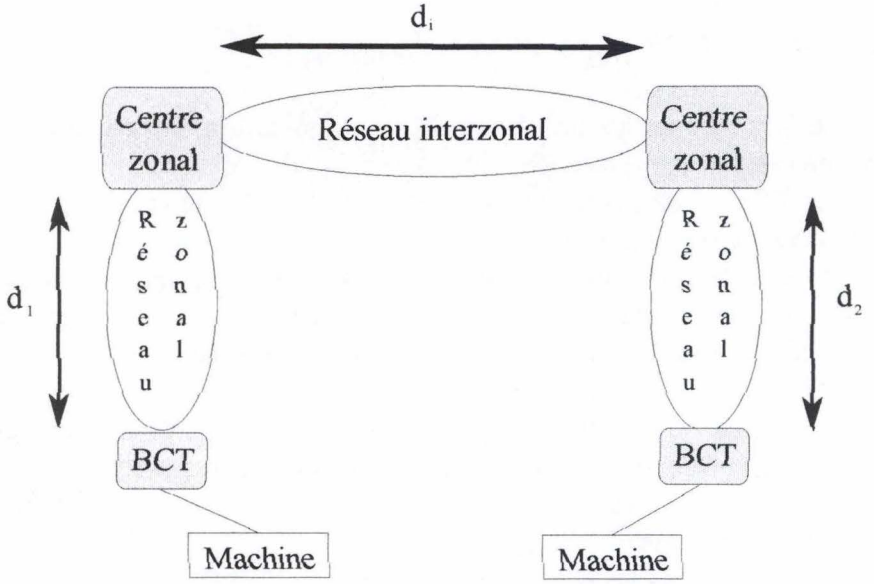

Figure 2.4 : Réseau à grande distance

Lorsque deux sites se trouvent à l'intérieur d'une zone, la partie interzonale n'est pas considérée (*Figure 2.5*); le coût n'est composé que de la partie zonale et du coût d'accès.

De même, si la ligne doit relier deux machines à l'intérieur du réseau local, les machines sont directement liées au même BCT. Dans ce cas, le coût se limite au coût d'accès.

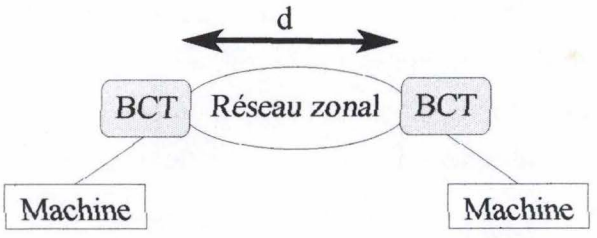

Figure 2.5 : Réseau zonal

Pour le cas d'un réseau ayant des machines dans des zones différentes, les tarifs d'une ligne interzonale se composent des éléments suivants (formule (2.3.)) :

- 2 x un montant d'accès au service "ligne louée" (2 x A(c));
- la partie zonale par extrémité : un montant fixe zonal (Fz(c)) + un montant proportionnel à la distance à vol d'oiseau entre machine et centre de zone exprimée en km (Vz(c) x d);
- la partie interzonale : un montant fixe interzonal (Fi(c)) + un montant proportionnel à la distance à vol d'oiseau entre centres de zone exprimée en km (Vi(c) x d);

ce qui donne pour l'ensemble de la ligne (de capacité c) de l'exemple de la *Figure 2.4* un coût mensuel :

$$C_{com}(c) = \underbrace{[2 \times A(c)]}_{\text{accès}} + \underbrace{[2 \times Fz(c) + Vz(c) \times d_1 + Vz(c) \times d_2]}_{\text{partie zonale}} + \underbrace{[Fi(c) + Vi(c) \times d_i]}_{\text{partie interzonale}} \qquad (2.3.)$$

Remarquons que si l'une des extrémités de la ligne interzonale est dans le réseau local du centre de zone, le montant pour la partie zonale de cette extrémité ne doit pas être comptabilisé.

Lorsqu'il s'agit d'un réseau zonal (*Figure 2.5*), le coût mensuel est donné par les éléments suivants :

- 2 x un montant d'accès au service "ligne louée" (2 x A(c));
- la partie zonale par extrémité : un montant fixe zonal (Fz(c)) + un montant proportionnel à la distance à vol d'oiseau entre machine et centre de zone exprimée en km (Vz(c) x d);

ce qui donne pour l'ensemble de la ligne (de capacité c) de la *Figure 2.5* :

$$C_{com}(c) = \underbrace{[2 \times A(c)]}_{\text{accès}} + \underbrace{[Fz(c) + Vz(c) \times d]}_{\text{partie zonale}} \qquad (2.4.)$$

Si deux machines sont reliées à l'intérieur d'un même réseau local (donc par à un même BCT), ce coût est uniquement composé des deux montants d'accès (A(c)) au service "ligne louée".

Ces montants sont classés en fonction de la capacité c de la ligne par tranches de 64 Kbps. La plage de capacités disponibles s'étend de 64 Kbps à 2 Mbps (par multiples de 64 Kbps).

Matrice des coûts

Comme déjà mentionné plus haut, l'étude des coûts de communication donne lieu à une optimisation du coût global du réseau. Ces coûts dépendent des capacités des lignes et peuvent être exprimés dans la matrice des coûts.

Calculons, à titre d'exemple, le coût mensuel de la ligne Bruges-Gand du réseau de la *Figure 2.1.* La capacité de cette ligne est de 466,6 Kbps (*Tableau 2.7*) et la capacité réalisable supérieure que le fournisseur met à la disposition de ses clients est de 512 Kbps (*Tableau 2.8*). Le tarif d'une ligne ayant une telle capacité se présente actuellement sous la forme suivante (*Tableau 2.9*) :

Capacité (en Kbps)	A (en FB)	Fz (en FB)	Vz (en FB/km)	Fi (en FB)	Vi (en FB/km)
64	4.177	6.116	21	11.310	125
128	4.733	6.331	32	14.260	163
192	5.266	6.545	44	17.211	200
256	5.843	6.760	55	20.161	238
320	6.397	6.974	67	23.111	276
384	6.950	7.189	78	26.062	313
448	7.503	7.403	89	29.012	351
512	8.055	7.616	101	31.962	389
576	8.606	7.832	112	34.913	427
...					
2048	20.000	12.337	352	96.870	1.218

Tableau 2.9 : Tarif d'une ligne[1]

Comme Gand et Bruges sont des centres d'une zone, la ligne ne passe pas par le réseau zonal; les machines sont directement reliées aux centres de zone. Le coût de l'abonnement d'une ligne de 512 Kbps entre Bruges et Gand n'est donc composé que du coût d'accès et du coût interzonal entre la zone de Bruges et celle de Gand.

Sachant que la distance (d_i) entre Gand et Bruges est d'environ 45 km, la formule du coût mensuel de communication (formule (2.3.)) s'exprime de la manière suivante :

$$C_{com} = [2 \times 8.055 \text{ FB}] + [31.962 \text{ FB} + 389 \text{ FB/km} \times 45 \text{ km}]$$

ce qui donne un coût mensuel pour la ligne Bruges-Gand de 65.577 FB.

On peut effectuer les mêmes calculs pour les autres lignes du réseau de la *Figure 2.1* et on trouve finalement la matrice des coûts (*Tableau 2.10*).

[1] Extrait de la tarification des lignes louées par Belgacom (tarifs du 01/07/1994)

Matrice des coûts (en FB/mois)								
Machine / Machine	Bruges	Anvers	Liège	Namur	Mons	Gand	Louvain	Nivelles
Bruges	0	0	0	0	0	65577	0	0
Anvers	0	0	0	0	0	0	67522	0
Liège	0	0	0	0	0	0	93295	0
Namur	0	0	0	0	0	0	0	71340
Mons	0	0	0	0	0	0	0	61687
Gand	65577	0	0	0	0	0	56605	42743
Louvain	0	67522	93295	0	0	56605	0	67522
Nivelles	0	0	0	71340	61687	42743	67522	0

Tableau 2.10 : Matrice des coûts

Chaque élément C_{ij} de la matrice des coûts C représente le coût mensuel des lignes entre la machine i et la machine j. Cette matrice des coûts possède évidemment les mêmes propriétés que la matrice des connexions : elle est symétrique et les coûts sont nuls pour les machines qui ne sont pas reliées directement entre elles.

En généralisant, on peut dire que le coût mensuel du réseau entier s'écrit :

$$C_{réseau} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} C_{ij} + C_{maint.} \qquad (2.5.)$$

où C_{ij} est l'élément (i,j) de la matrice des coûts, n le nombre de machines et $C_{maint.}$ représente les coûts mensuels de maintenance et de l'entretien du réseau.

2.3. Performances d'un réseau

A coté des coûts, il y a un deuxième critère qui influence la conception d'un réseau de communication : la performance d'un réseau.

Un réseau doit posséder des performances raisonnables. Il est clair que plus la qualité du réseau est élevée, plus le coût du réseau le sera aussi. Cette notion de performance influence donc le coût du réseau.

Comme on l'a dit plus haut, l'étude des files d'attente a montré qu'un taux d'occupation de tout au plus 60% garantit le bon déroulement des transmissions de messages. On considère que cette borne supérieure garantit que le réseau ne sera jamais surchargé (sinon le moins souvent possible). Avec ce taux d'occupation, les temps d'attente entre noeuds restent acceptables. Rappelons que ce taux d'occupation agit sur le coût du réseau (cf. formule (2.1.)). Nous allons nous restreindre au taux d'occupation comme critère de performance. Les études de performance à l'aide de simulations sont des sujets à part entière.

Pour que chaque utilisateur puisse avoir une idée plus précise des performances du réseau, on peut calculer le temps de réponse sur chaque liaison. Ceci se fait souvent par des calculs analytiques complexes et sophistiqués. Ces calculs font appel à des techniques de la théorie des processus stochastiques. Sans entrer dans les détails de ces techniques, ce travail se contentera de formuler une approche assez intuitive du temps de transmission.

Comme chaque ligne du réseau a une capacité limitée, il y a attente à l'entrée de ces lignes s'il y a trop de messages (paquets). Chaque machine doit donc posséder une file d'attente qui stocke les paquets lorsque la ligne est occupée et qui les émet dès que la ligne est libre. Comme des systèmes de files d'attente de taille finie sont très difficiles à analyser, on va supposer que la taille des files est illimitée. Il faut également déterminer selon quelle stratégie on choisit le prochain "client" à servir. La stratégie utilisée dans le cadre de ce mémoire est FIFO (First In - First Out), puisqu'elle donne lieu à des formules simples.

Or, si dans un réseau à commutation par paquets les paquets sont de taille fixe, les machines du réseau ont une file d'attente de type M/C/1 (le 1 indique qu'il n'y a qu'une file d'attente). Ceci implique que l'on suppose que la distribution des paquets arrivant dans la file d'attente (de la machine) vérifie une loi poissonienne, c'est-à-dire qu'il faut que l'arrivée des paquets soit indépendante (ceci est indiqué par le M dans M/C/1). Si l'arrivée des paquets est poissonienne, la distribution des temps inter-arrivée est exponentielle. Bien que cette hypothèse ne soit qu'une approximation, elle est raisonnable et est faite couramment car elle simplifie considérablement les formules des files d'attente.

En plus, on a supposé que les paquets sont de taille fixe. La distribution du temps de service est donc constante (indiqué par un C dans M/C/1). Le temps de traitement dans un noeud est supposé nul.

Le temps de service d'un paquet - c'est-à-dire le temps pour émettre un paquet sur une ligne - s'exprime comme la longueur du paquet l (en bits) divisée par la capacité de la ligne C (exprimée en bps) :

$$t_{serv.} = \frac{l}{C} \qquad (2.6.)$$

où $t_{serv.}$ est le temps de service (en s)

D'après les formules classiques des files d'attente, le nombre moyen de paquets (Np) de taille fixe dans la file d'attente peut s'écrire comme le rapport suivant :

$$Np = \frac{\rho}{2(1-\rho)} \qquad (2.7.)$$

La *Figure 2.6* montre l'influence du taux d'occupation sur le nombre moyen de messages en attente. On voit que le nombre de paquets en attente tend vers l'infini lorsque le taux d'occupation tend vers 1.

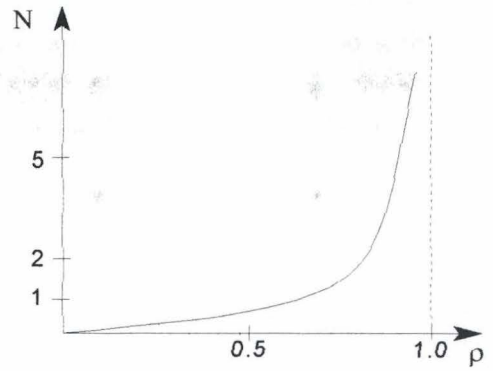

Figure 2.6 : Relation entre le taux d'occupation et le nombre de messages en attente

D'autre part, le temps d'attente moyen d'un paquet de taille fixe dans une file d'attente de type M/C/1 s'exprime de la manière suivante :

$$\bar{t}_{attente} = \frac{\rho}{2(1-\rho)} \cdot t_{serv.} \qquad (2.8.)$$

où $\bar{t}_{attente}$ est le temps d'attente moyen (en s)
$t_{serv.}$ est le temps de service (en s)

Le temps nécessaire à la transmission d'un paquet sur une ligne, c'est-à-dire le temps de transfert d'un bout à l'autre d'une ligne, se décompose en trois parties : le temps d'attente dans la file d'attente, le temps qu'il faut pour transmettre le paquet (le temps de service) et le temps de propagation sur la ligne. Ce temps de propagation est le temps qu'un paquet prend pour se propager sur la ligne :

$$t_{propag.} = \frac{d}{v_{propag.}} \qquad (2.9.)$$

où $t_{propag.}$ est le temps de propagation (en s)
d est la longueur de la ligne (en km)
$v_{propag.}$ est la vitesse de propagation du paquet (± 200.000 km/s)

Le temps de transfert moyen d'un paquet s'écrit finalement comme la somme du temps d'attente moyen, du temps de service et du temps de propagation :

$$\bar{t}_{trans.} = \bar{t}_{attente} + t_{serv.} + t_{propag.} \qquad (2.10.)$$

On peut, à titre d'exemple, calculer le temps de transfert de la ligne Bruges-Gand de la *Figure 2.1.* Calculons d'abord le temps de service : sachant que la capacité réalisable de la ligne est de 512 Kbps et que les paquets ont une taille fixe de 250 bytes, le temps de service s'écrit (formule (2.6.)) :

$$t_{serv.} = \frac{250 \cdot 8 \text{ bits}}{512 \text{ Kbps}} = 3{,}91 \text{ ms}$$

Le temps de propagation sur cette ligne est donné par le temps qu'un paquet prend pour se propager sur la ligne de 45 km entre Bruges et Gand :

$$t_{propag.} = \frac{45 \text{ km}}{200.000 \text{ km / s}} = 0{,}23 \text{ ms}$$

Etant donné que le temps d'attente dépend du taux d'occupation et donc du trafic sur la ligne, nous allons calculer ce temps dans les deux sens de transmission : d'abord de Bruges vers Gand et puis dans le sens opposé.

Comme on doit connaître le taux d'occupation effectif sur la ligne Bruges→Gand, calculons ρ :

$$\rho = \frac{130 \text{ paq./sec.} \cdot 2000 \text{ bits/paq.}}{512.000 \text{ bps}} = \frac{260 \text{ Kbps}}{512 \text{ Kbps}} = 0{,}508$$

La formule (2.8.) permet alors de trouver le temps d'attente moyen dans l'hôte de Bruges pour pouvoir accéder à la ligne Bruges-Gand :

$$\bar{t}_{attente} = \frac{0{,}508}{2 \cdot (1 - 0{,}508)} \cdot 3{,}91 \text{ ms} = 2{,}02 \text{ ms}$$

Le temps d'attente moyen pour accéder à la ligne Bruges-Gand dans le noeud de Gand se calcule de la même manière :

$$\rho = \frac{\lambda \cdot 1}{C} = \frac{140 \text{ paq./sec.} \cdot 2000 \text{ bits/paq.}}{512.000 \text{ bps}} = \frac{280 \text{ Kbps}}{512 \text{ Kbps}} = 0{,}547$$

et

$$\bar{t}_{attente} = \frac{0{,}547}{2 \cdot (1 - 0{,}547)} \cdot 3{,}91 \text{ ms} = 2{,}36 \text{ ms}$$

Finalement, le temps de transfert moyen d'un paquet Bruges→Gand s'écrit comme la somme des trois temps trouvés précédemment :

$$\bar{t}_{trans.} = 3{,}91 \text{ ms} + 2{,}02 \text{ ms} + 0{,}23 \text{ ms} = 6{,}16 \text{ ms}$$

Dans l'autre direction, ce temps vaut :

$$\bar{t}_{trans.} = 3{,}91 \text{ ms} + 2{,}36 \text{ ms} + 0{,}23 \text{ ms} = 6{,}50 \text{ ms}$$

On peut effectuer ces calculs pour toutes les lignes du réseau. Les temps de transfert moyens résultants sont alors notés dans la matrice des temps de transfert moyen d'un paquet sur une ligne (*Tableau 2.11*). Comme le temps d'attente est différent dans chaque noeud, cette matrice n'est pas symétrique.

Matrice des temps de transfert d'un paquet sur une ligne (en ms)								
Machine / Machine	Bruges	Anvers	Liège	Namur	Mons	Gand	Louvain	Nivelles
Bruges	0	0	0	0	0	6,16	0	0
Anvers	0	0	0	0	0	0	6,88	0
Liège	0	0	0	0	0	0	5,00	0
Namur	0	0	0	0	0	0	0	5,56
Mons	0	0	0	0	0	0	0	6,83
Gand	6,50	0	0	0	0	0	10,61	14,45
Louvain	0	6,47	5,45	0	0	10,61	0	6,13
Nivelles	0	0	0	5,84	5,79	16,39	6,47	0

Tableau 2.11 : Matrice des temps de transfert d'un paquet sur une ligne

Pour trouver le temps de transfert moyen sur une liaison (donc entre deux hôtes), il suffit de faire la somme des temps de transfert moyens de chaque ligne composant la liaison. Les lignes que chaque liaison emprunte sont données dans le *Tableau 2.5*. Pour la liaison Bruges-Namur, par exemple, il suffit de faire la somme des temps de transfert des lignes Bruges-Gand, Gand-Nivelles et Nivelles-Namur :

$$T_{\text{liaison Br-Na}} = 6,16 + 14,45 + 5,84 = 26,45 \text{ ms}$$

Les temps de transfert de toutes les liaisons se résument dans la matrice du *Tableau 2.12*.

Matrice des temps de transfert (en ms) - liaisons					
Machine / Machine	Bruges	Anvers	Liège	Namur	Mons
Bruges	0	23,24	22,22	26,45	26,40
Anvers	23,99	0	12,33	18,85	18,80
Liège	22,11	11,47	0	16,97	16,92
Namur	28,45	18,50	17,48	0	11,35
Mons	29,72	19,77	18,75	12,67	0

Tableau 2.12 : Matrice des temps de transfert d'un paquet sur une liaison

2.4. Application des concepts

Nous allons maintenant, à titre d'exemple, appliquer les concepts évoqués ci-dessus à un réseau d'une taille plus importante. Dans ce réseau, on représentera un noeud (routeur) et un hôte directement attaché au noeud par une seule machine. Ce sera donc une machine pouvant router et émettre des paquets. En plus, le routage va se compliquer par rapport à l'exemple de la *Figure 2.1*, puisque les paquets passeront plusieurs noeuds pour arriver à la destination. Nous allons donc parcourir toutes les étapes de la conception d'un réseau expliquées dans les sections précédentes.

Le réseau

Le réseau envisagé (*Figure 2.7*) est un réseau privé à commutation par paquets basé sur des lignes louées. Il est composé de 17 hôtes (les carrés) et 5 noeuds et sera représenté par 12 hôtes et 5 machines "noeud/hôte".

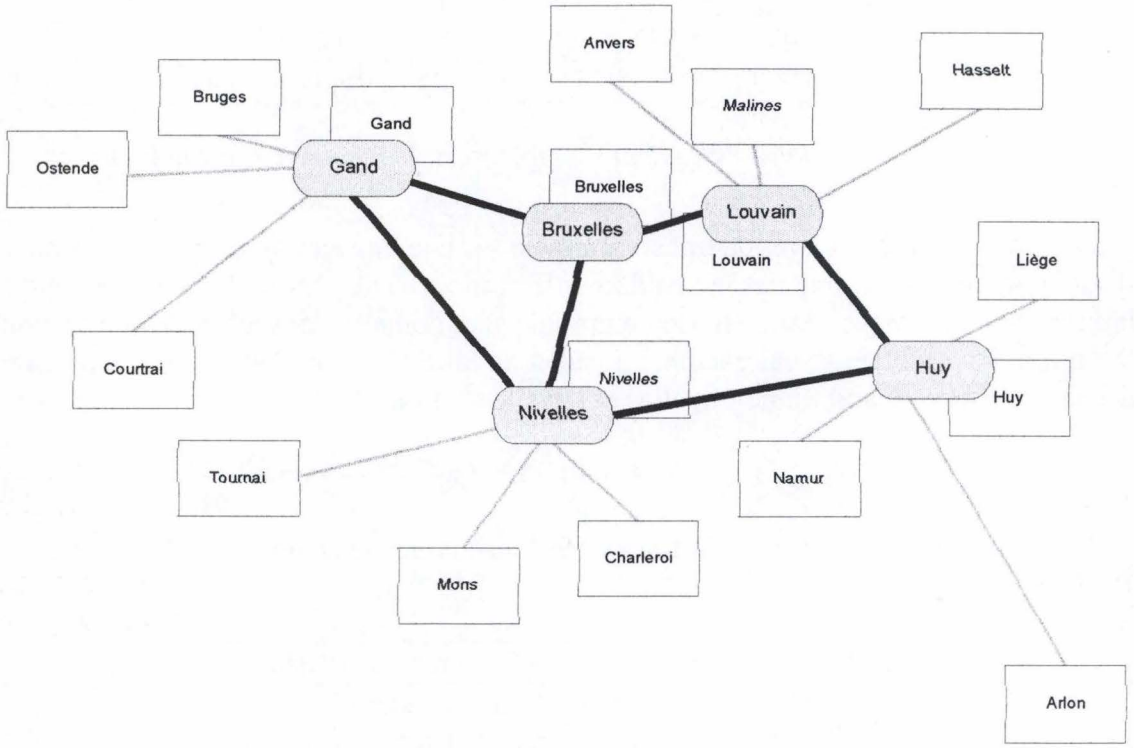

Figure 2.7 : Exemple d'un réseau de communication

Pour pouvoir calculer le coût et les performances, il faut que l'emplacement des machines (hôtes et noeuds), les distances entre machines ainsi que la communication entre machines (donné par la matrice de communication) soient connus. Le routage est fixé dès le début.

Matrice de communication

Avant d'entamer commencer le calcul des capacités et des coûts, il faut analyser les besoins de communication. Les résultats de cette analyse sont résumés dans la matrice de communication. Comme les communications entre hôtes sont indépendantes les unes des autres, cette matrice n'est pas symétrique. Le *Tableau 2.13* montre la matrice de communication de l'exemple en question (*Figure 2.7*).

Matrice de communication (en paquets/sec.)																	
Machine / Machine	Bxl	Lv.	Huy	Niv.	Gd	An.	Mal.	Has.	Lg.	Arl.	Nr.	Chl.	Mns	Trn.	Crt.	Ost.	Brg.
Bruxelles	0	40	25	25	35	30	20	20	35	15	30	30	20	15	20	25	30
Louvain	40	0	25	20	30	30	20	25	35	10	25	25	20	15	10	20	20
Huy	25	20	0	20	40	15	15	20	30	20	25	20	20	10	15	10	15
Nivelles	30	25	30	0	25	10	10	15	25	15	30	35	30	25	15	10	15
Gand	35	35	15	15	0	30	20	15	20	10	20	20	15	15	20	35	40
Anvers	20	30	15	15	25	0	30	25	20	10	20	20	15	10	15	25	30
Malines	15	25	20	10	20	30	0	20	15	5	15	10	5	5	10	15	20
Hasselt	10	15	10	10	20	25	25	0	15	5	15	15	10	5	5	20	20
Liège	30	20	30	25	25	25	20	25	0	25	30	25	20	20	15	15	25
Arlon	15	15	35	20	10	10	5	10	25	0	20	15	10	10	5	5	5
Namur	25	20	25	30	20	15	10	10	25	20	0	35	25	20	10	10	20
Charleroi	35	20	15	30	25	15	15	15	25	15	35	0	30	25	15	15	25
Mons	25	20	15	25	15	10	10	5	20	10	25	35	0	20	10	5	30
Tournai	20	15	10	20	15	10	5	10	15	10	20	25	25	0	10	10	40
Courtrai	15	10	10	15	25	15	5	5	10	5	10	20	15	15	0	25	30
Ostende	20	10	5	5	30	20	10	15	5	5	20	10	10	10	25	0	40
Bruges	35	20	10	15	40	30	20	15	15	10	30	20	10	15	30	35	0

Tableau 2.13 : Matrice de communication

Les éléments de la matrice représentent le nombre de paquets par seconde passant d'un hôte source (indice des lignes) vers un hôte destination (indice des colonnes). Dans cet exemple, ces paquets ont une taille fixe de 250 bytes (2000 bits). Le trafic moyen de Namur → Liège, par exemple, est de 25 paquets par seconde, autrement dit de 50 Kbps (25 paq./sec. · 2000 bits/paq.), tandis que le trafic de Liège → Namur est de 30 paquets par seconde (60 Kbps).

Matrice de routage

Les paquets qui sont envoyés d'un hôte à un autre peuvent prendre différents chemins. Ainsi, tous les messages qui vont par exemple de Bruxelles vers Huy peuvent passer ou bien par Nivelles ou bien par Louvain (*Figure 2.8*).

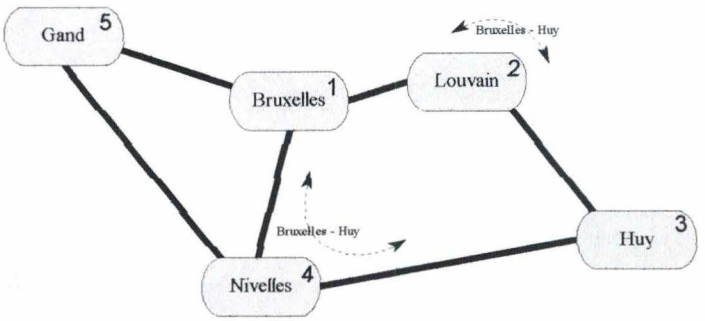

Figure 2.8 : Routage

Il faut donc choisir une politique de routage. La politique adoptée dans le cadre de ce travail est celle du routage fixe qui a été choisi pour minimiser la distance à parcourir. Le routage fixe définit chaque chemin de façon unique : les paquets passent toujours par les mêmes noeuds. Pour que chaque noeud sache vers quel noeud il doit envoyer les paquets venant d'une certaine machine, les noeuds possèdent des tables de routage (*Tableau 2.14*). Ces tables sont initialisées au début de l'installation du réseau et leur contenu est stable. Ce contenu est stable dans le sens où il reste inchangé tant qu'aucune ligne ne tombe en panne.

On peut regrouper les tables de routage de chaque noeud dans la matrice de routage. Chaque ligne de la matrice représente la table de routage du noeud indiqué au début de la ligne.

Matrice de routage					
Machine / Machine	Bruxelles	Louvain	Huy	Nivelles	Gand
Bruxelles	/	d	Louvain	d	d
Louvain	d	/	d	Bruxelles	Bruxelles
Huy	Louvain	d	/	d	Nivelles
Nivelles	d	Bruxelles	d	/	d
Gand	d	Bruxelles	Nivelles	d	/

Tableau 2.14 : Matrice de routage

L'élément R_{ij} de la matrice de routage R désigne le noeud vers lequel le noeud i doit envoyer les paquets pour joindre le noeud j. Le *d* signifie que les paquets seront envoyés directement vers la destination sans passer par un noeud intermédiaire. Ainsi, pour l'exemple de la *Figure 2.7*, les paquets qui vont de Bruxelles vers Huy sont routés par le noeud de Louvain et les paquets de Bruxelles destinés à Gand sont directement envoyés sur la ligne vers Gand.

Remarquons que la matrice de routage n'est pas nécessairement symétrique. Comme le routage est fixe, ceci dépend du choix du concepteur de réseau. Dans notre cas, elle sera toujours symétrique.

Matrice du trafic par ligne

En tenant compte de la matrice de routage ci-dessus et sachant que le trafic moyen pour chaque ligne est constitué par le trafic moyen des liaisons empruntant chacune des lignes, on trouvera la matrice du trafic par ligne (*Tableau 2.15*).

Le trafic moyen d'une ligne hôte-noeud est la somme des trafics de toute liaison entre cet hôte et tous les autres hôtes. Il est assez facile à trouver : le trafic hôte→noeud se calcule en faisant la somme de la ligne correspondante dans la matrice de communication. La somme de toute la colonne donne le trafic moyen noeud→hôte correspondant. Pour la ligne Huy-Liège , par exemple, le trafic moyen Liège→Huy est de 375 paq./sec. et le trafic opposé de 335 paq./sec.

Pour calculer le trafic moyen d'une ligne entre noeuds, il faut tenir compte de la matrice de routage (*Tableau 2.14*). Le trafic sera la somme du trafic des liaisons empruntant la ligne. Dans le cas de la ligne Bruxelles-Nivelles, la matrice de routage nous indique que les paquets entre les noeuds de Bruxelles et de Louvain et le noeud de Nivelles sont les seuls à passer sur la ligne. Il faut donc faire la somme des trafics des liaisons entre hôtes reliés à Bruxelles et Louvain et les hôtes reliés à Nivelles. Dans la direction Bruxelles→Nivelles, le trafic moyen sur la ligne vaut alors 300 paq./sec. et dans le sens opposé 320 paq./sec.

Matrice du trafic par ligne(en paquets/sec)																	
Machine / Machine	Bxl	Lv.	Huy	Niv.	Gd	An.	Mal.	Has.	Lg.	Arl.	Nr.	Chl.	Mns	Trn.	Crt.	Ost.	Brg.
Bruxelles	0	600	0	300	380	0	0	0	0	0	0	0	0	0	0	0	0
Louvain	330	0	365	0	0	320	240	250	0	0	0	0	0	0	0	0	0
Huy	0	350	0	315	0	0	0	0	335	280	370	0	0	0	0	0	0
Nivelles	320	0	570	0	425	0	0	0	0	0	0	360	280	235	0	0	0
Gand	335	0	0	280	0	0	0	0	0	0	0	0	0	0	230	290	405
Anvers	0	325	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Malines	0	240	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hasselt	0	225	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Liège	0	0	375	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Arlon	0	0	215	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Namur	0	0	320	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Charleroi	0	0	0	355	0	0	0	0	0	0	0	0	0	0	0	0	0
Mons	0	0	0	280	0	0	0	0	0	0	0	0	0	0	0	0	0
Tournai	0	0	0	260	0	0	0	0	0	0	0	0	0	0	0	0	0
Courtrai	0	0	0	0	230	0	0	0	0	0	0	0	0	0	0	0	0
Ostende	0	0	0	0	240	0	0	0	0	0	0	0	0	0	0	0	0
Bruges	0	0	0	0	350	0	0	0	0	0	0	0	0	0	0	0	0

Tableau 2.15 : Matrice du trafic par ligne

Matrice des connexions

En appliquant la formule (2.1.) (page 14) au plus grand trafic des deux directions d'une ligne, on trouvera la matrice des connexions (*Tableau 2.16*). Le trafic moyen sur la ligne Bruxelles - Louvain par exemple est de 2 Mbps. Cette matrice vérifie les propriétés décrites dans la section 2.1. Rappelons que le taux d'occupation désiré (donc maximal) ρ est fixé à 60 %.

Matrice des connexions (en Kbps)																	
Machine / Machine	Bxl	Lv.	Huy	Niv.	Gd	An.	Mal.	Has.	Lg.	Arl.	Nr.	Chl.	Mns	Trn.	Crt.	Ost.	Brg.
Bruxelles	0	2000	0	1066	1266	0	0	0	0	0	0	0	0	0	0	0	0
Louvain	2000	0	1166	0	0	1083	800	833	0	0	0	0	0	0	0	0	0
Huy	0	1166	0	1900	0	0	0	0	1250	716	1233	0	0	0	0	0	0
Nivelles	1066	0	1900	0	1416	0	0	0	0	0	0	1200	933	866	0	0	0
Gand	1266	0	0	1416	0	0	0	0	0	0	0	0	0	0	766	933	1350
Anvers	0	1083	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Malines	0	800	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hasselt	0	833	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Liège	0	0	1250	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Arlon	0	0	716	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Namur	0	0	1233	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Charleroi	0	0	0	1200	0	0	0	0	0	0	0	0	0	0	0	0	0
Mons	0	0	0	933	0	0	0	0	0	0	0	0	0	0	0	0	0
Tournai	0	0	0	866	0	0	0	0	0	0	0	0	0	0	0	0	0
Courtrai	0	0	0	0	766	0	0	0	0	0	0	0	0	0	0	0	0
Ostende	0	0	0	0	933	0	0	0	0	0	0	0	0	0	0	0	0
Bruges	0	0	0	0	1350	0	0	0	0	0	0	0	0	0	0	0	0

Tableau 2.16 : Matrice des connexions

A partir de cette matrice, on peut trouver la capacité des lignes à installer et construire la matrice des capacités réalisables (aussi appelée matrice des lignes).

Matrice des lignes (en Kbps)																	
Machine / Machine	Bxl	Lv.	Huy	Niv.	Gd	An.	Mal.	Has.	Lg.	Arl.	Nr.	Chl.	Mns	Trn.	Crt.	Ost.	Brg.
Bruxelles	0	2048	0	1088	1280	0	0	0	0	0	0	0	0	0	0	0	0
Louvain	2048	0	1216	0	0	1088	832	896	0	0	0	0	0	0	0	0	0
Huy	0	1216	0	1920	0	0	0	0	1280	768	1280	0	0	0	0	0	0
Nivelles	1088	0	1920	0	1472	0	0	0	0	0	0	1216	960	896	0	0	0
Gand	1280	0	0	1472	0	0	0	0	0	0	0	0	0	0	768	960	1408
Anvers	0	1088	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Malines	0	832	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hasselt	0	896	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Liège	0	0	1280	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Arlon	0	0	768	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Namur	0	0	1280	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Charleroi	0	0	0	1216	0	0	0	0	0	0	0	0	0	0	0	0	0
Mons	0	0	0	960	0	0	0	0	0	0	0	0	0	0	0	0	0
Tournai	0	0	0	896	0	0	0	0	0	0	0	0	0	0	0	0	0
Courtrai	0	0	0	0	768	0	0	0	0	0	0	0	0	0	0	0	0
Ostende	0	0	0	0	960	0	0	0	0	0	0	0	0	0	0	0	0
Bruges	0	0	0	0	1408	0	0	0	0	0	0	0	0	0	0	0	0

Tableau 2.17 : Matrice des lignes

La matrice des lignes (*Tableau 2.17*) s'obtient en arrondissant la capacité des lignes au multiple de 64 Kbps supérieur le plus proche. Elle donne la capacité réelle de chaque ligne physique.

Matrice des coûts

Comme tous les hôtes et noeuds se trouvent dans des centres zonaux, il n'existe pas de partie zonale. Lorsqu'on applique les tarifs aux éléments de la matrice des lignes en utilisant la formule (2.3.), on trouve les coûts de transmission mensuels. Ils se résument dans la matrice des coûts du *Tableau 2.18.*

Calculons - à titre d'exemple - le coût de la ligne Bruxelles-Louvain. La capacité de cette ligne est 2 Mbps et sa longueur de 25 km. En appliquant le tarif de la ligne (*Tableau 2.9*) à la formule (2.3.), on obtient les équations suivantes :

$$C_{Bxl\text{-}Lv}(2\ Mbps) = 2 \times A(2Mbps) + Fi(2Mbps) + Vi(2\ Mbps) \times 25\ km$$
$$= 40.000\ FB + 96.870\ FB + 1.218\ FB/km \times 25\ km$$
$$= 167.320\ FB$$

Matrice des coûts (en milliers de FB/mois)																	
Machine / Machine	Bxl	Lv.	Huy	Niv.	Gd	An.	Mal.	Has.	Lg.	Arl.	Nr.	Chl.	Mns	Trn.	Crt.	Ost.	Brg.
Bruxelles	0	167	0	88	96	0	0	0	0	0	0	0	0	0	0	0	0
Louvain	167	0	145	0	0	121	80	106	0	0	0	0	0	0	0	0	0
Huy	0	145	0	222	0	0	0	0	122	124	122	0	0	0	0	0	0
Nivelles	106	0	222	0	180	0	0	0	0	0	0	113	99	118	0	0	0
Gand	147	0	0	180	0	0	0	0	0	0	0	0	0	0	91	122	146
Anvers	0	121	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Malines	0	80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hasselt	0	106	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Liège	0	0	122	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Arlon	0	0	124	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Namur	0	0	122	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Charleroi	0	0	0	113	0	0	0	0	0	0	0	0	0	0	0	0	0
Mons	0	0	0	99	0	0	0	0	0	0	0	0	0	0	0	0	0
Tournai	0	0	0	118	0	0	0	0	0	0	0	0	0	0	0	0	0
Courtrai	0	0	0	0	91	0	0	0	0	0	0	0	0	0	0	0	0
Ostende	0	0	0	0	122	0	0	0	0	0	0	0	0	0	0	0	0
Bruges	0	0	0	0	146	0	0	0	0	0	0	0	0	0	0	0	0

Tableau 2.18 : Matrice des coûts

Cette matrice permet de trouver le coût de communication de tout le réseau. Dans cet exemple, le coût total s'élève à 4,6 millions de FB par mois.

Matrice des temps de réponse

Comme déjà mentionné plus haut, le temps de réponse est un des critères de performance. Cherchons, à titre d'exemple le temps de transfert moyen entre Ostende et Arlon.

Un paquet d'Ostende à destination d'Arlon passe d'abord par le noeud situé à Gand. Ce noeud cherche dans sa table de routage (cf. *Tableau 2.14*) le noeud situé à Huy (= le noeud d'entrée de l'hôte d'Arlon). En se basant sur l'entrée dans la table, il envoie le paquet vers le noeud situé à Nivelles. Ce dernier dirige le paquet vers Huy en

procédant de la même manière. Le noeud à Huy transmet le paquet finalement à l'hôte de destination Arlon. La liaison Ostende → Arlon est donc composée des lignes Ostende-Gand, Gand-Nivelles, Nivelles-Huy et Huy-Arlon. On ne va ici que calculer le temps de transfert de l'une de ces lignes : la ligne Gand-Nivelles.

Pour trouver le temps de transfert moyen d'un paquet sur une ligne, il faut calculer le temps de service, le temps d'attente moyen et le temps de propagation (formule (2.10.)).

Le temps de service s'écrit comme le rapport entre la longueur d'un paquet (en bits) et la capacité de la ligne (formule (2.6.)) :

$$t_{serv.} = \frac{250 \cdot 8 \text{ bits}}{1472 \text{ kbps}} = 1{,}36 \text{ ms}$$

Pour pouvoir calculer le temps d'attente, il faut connaître le taux d'occupation effectif de la ligne :

$$\rho = \frac{\lambda \cdot 1}{C} = \frac{425 \text{ paq./sec.} \cdot 2000 \text{ bits / paq.}}{1472 \text{ Kbps}} = 0{,}577$$

Le temps d'attente moyen dans le noeud de Gand se calcule par la formule (2.8.):

$$\bar{t}_{attente} = \frac{0{,}577}{2 \cdot (1 - 0{,}577)} \cdot 1{,}36 \text{ ms} = 0{,}93 \text{ ms}$$

Il reste à calculer le temps de propagation qu'il faut pour qu'un paquet parcoure la distance de 75 km entre Gand et Nivelles. Etant donné la vitesse de propagation de 200.000 km/s, ce temps s'exprime par (formule (2.9.)) :

$$t_{propag.} = \frac{75 \text{ km}}{200.000 \text{ km / s}} = 0{,}38 \text{ ms}$$

Finalement, le temps de transfert moyen sur une ligne s'écrit comme la somme de ces trois temps :

$$\bar{t}_{trans.} = 1{,}36 \text{ ms} + 0{,}93 \text{ ms} + 0{,}38 \text{ ms} = 2{,}67 \text{ ms}$$

Les temps de transfert des autres lignes du réseau se calculent d'une façon semblable et sont résumés dans la matrice du temps de transfert d'un paquet sur une ligne (*Tableau 2.19*) :

Matrice du temps de transfert d'un paquet sur les lignes (en ms)																	
Machine / Machine	Bxl	Lv.	Huy	Niv.	Gd	An.	Mal.	Has.	Lg.	Arl.	Nr.	Chl.	Mns	Trn.	Crt.	Ost.	Brg.
Bruxelles	0	1.8	0	3.1	2.9	0	0	0	0	0	0	0	0	0	0	0	0
Louvain	1.3	0	2.8	0	0	3.4	4.1	3.8	0	0	0	0	0	0	0	0	0
Huy	0	2.8	0	1.6	0	0	0	0	2.6	4	2.8	0	0	0	0	0	0
Nivelles	3.3	0	2.1	0	2.7	0	0	0	0	0	0	2.9	3.7	3.8	0	0	0
Gand	2.7	0	0	2.2	0	0	0	0	0	0	0	0	0	0	4.7	3.5	2.6
Anvers	0	3.4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Malines	0	4.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hasselt	0	3.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Liège	0	0	2.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Arlon	0	0	3.4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Namur	0	0	2.5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Charleroi	0	0	0	2.9	0	0	0	0	0	0	0	0	0	0	0	0	0
Mons	0	0	0	3.7	0	0	0	0	0	0	0	0	0	0	0	0	0
Tournai	0	0	0	4.1	0	0	0	0	0	0	0	0	0	0	0	0	0
Courtrai	0	0	0	0	4.7	0	0	0	0	0	0	0	0	0	0	0	0
Ostende	0	0	0	0	3.1	0	0	0	0	0	0	0	0	0	0	0	0
Bruges	0	0	0	0	2.3	0	0	0	0	0	0	0	0	0	0	0	0

Tableau 2.19 : Matrice du temps de transfert d'un paquet sur les lignes

Pour trouver le temps de transfert moyen sur la liaison Ostende → Arlon, il suffit de faire la somme du temps de transfert des lignes qui composent la liaison :

$$\bar{t}_{trans./Ost.-Arlon} = \bar{t}_{trans./Ost.-Gand} + \bar{t}_{trans./Gand-Niv.} + \bar{t}_{trans./Niv.-Huy} + \bar{t}_{trans./Huy-Arlon}$$
$$= 3{,}1 \text{ ms} + 2{,}2 \text{ ms} + 2{,}1 \text{ ms} + 4{,}0 \text{ ms} = 11{,}4 \text{ ms}$$

Les temps de transfert moyens des autres liaisons sont visibles dans la matrice du *Tableau 2.20.*

Matrice du temps de transfert d'un paquet sur les liaisons (en ms)																	
Machine / Machine	Bxl	Lv.	Huy	Niv.	Gd	An.	Mal.	Has.	Lg.	Arl.	Nr.	Chl.	Mns	Trn.	Crt.	Ost.	Brg.
Bruxelles	0	1.8	4.1	3.1	2.9	4.7	5.4	4.9	6.9	7.5	6.6	6.2	7	7.4	7.4	5.8	5
Louvain	1.3	0	2.8	5.1	4.5	3.4	4.1	3.6	5.6	6.2	5.3	8	8.8	9.2	9.2	7.6	6.8
Huy	4.1	2.8	0	1.6	4.3	6.2	6.9	6.4	2.8	3.4	2.5	5	5.8	6.2	9	7.4	6.6
Nivelles	3.3	5.1	2.1	0	2.7	8.5	9.2	8.7	4.4	5	4.1	2.9	3.7	4.1	6.9	5.3	4.5
Gand	2.7	4.5	4.3	2.2	0	7.9	8.6	8.1	7.1	7.7	6.8	5.6	6.4	6.8	4.7	3.1	2.3
Anvers	4.7	3.4	6.2	8.5	7.9	0	7.5	7	9	9.6	8.7	12.9	13.7	14.1	12.6	11	10.2
Malines	5.4	4.1	6.9	9.2	8.6	7.5	0	7.7	9.7	10.3	9.4	13.6	14.4	14.8	13.3	11.7	10.9
Hasselt	4.9	3.6	6.4	8.7	8.1	7	7.7	0	9.4	10	9.1	13.3	14.1	14.5	13	11.4	10.6
Liège	6.9	5.6	2.8	4.4	7.1	9	9.7	9.4	0	6	5.1	7.6	8.4	8.8	11.6	10	9.2
Arlon	7.5	6.2	3.4	5	7.7	9.6	10.3	10	6	0	6.5	9	9.8	10.2	13	11.4	10.6
Namur	6.6	5.3	2.5	4.1	6.8	8.7	9.4	9.1	5.1	6.5	0	7.8	8.6	9	11.8	10.2	9.4
Charleroi	6.2	8	5	2.9	5.6	12.9	13.6	13.3	7.6	9	7.8	0	6.6	7	9.8	8.2	7.4
Mons	7	8.8	5.8	3.7	6.4	13.7	14.4	14.1	8.4	9.8	8.6	6.6	0	7.8	10.6	9	8.2
Tournai	7.4	9.2	6.2	4.1	6.8	14.1	14.8	14.5	8.8	10.2	9	7	7.8	0	10.7	9.1	8.3
Courtrai	7.4	9.2	9	6.9	4.7	12.6	13.3	13	11.6	13	11.8	9.8	10.6	10.7	0	7.8	7
Ostende	5.8	7.6	7.4	5.3	3.1	11	11.7	11.4	10	11.4	10.2	8.2	9	9.1	7.8	0	5.4
Bruges	5	6.8	6.6	4.5	2.3	10.2	10.9	10.6	9.2	10.6	9.4	7.4	8.2	8.3	7	5.4	0

Tableau 2.20 : Matrice du temps de transfert d'un paquet sur les liaisons

Pour avoir une idée globale de la performance du réseau, on peut calculer le temps de transfert moyen pour toutes les lignes et pour toutes les liaisons du réseau. Dans le cas de cet exemple, le temps de transfert en moyenne par ligne vaut 3,1 ms et le temps moyen par liaison vaut 7,7 ms.

Chapitre 3

Algorithmes de conception et d'optimisation d'un réseau

3.1. Problème de conception

La conception de réseaux de télécommunication consiste à déterminer la topologie d'un réseau. Pour obtenir cette topologie, il faut trouver le nombre de noeuds, leur localisation, le placement des lignes et la capacité de ces lignes. Le problème principal consiste à minimiser le coût mensuel total des lignes louées de la topologie trouvée tout en garantissant des performances raisonnables (comme le temps de réponse et la fiabilité). Dans la suite, on parlera de topologie optimale, lorsque la topologie vérifie les deux critères ci-dessus.

Il y a essentiellement deux types de topologies dans le monde des réseaux à grande distance. D'une part, les réseaux ayant une structure centralisée qui sont généralement des réseaux à un processeur ou noeud central. Des entreprises comme les banques, qui ont leur centre informatique à un endroit central, ont typiquement cette structure de réseaux. D'autre part, il y a les réseaux à structure distribuée n'ayant pas d'entité centrale. Ces derniers sont composés de plusieurs noeuds reliés entre eux. Nous allons présenter ces deux types de topologies et étudier le problème de leur conception.

Réseaux à structure centralisée

Les réseaux à structure centralisée ont tous une caractéristique commune : ils possèdent un processeur (ou un noeud) central. Tout le trafic du réseau doit aboutir ou passer par ce processeur. Chaque hôte doit être relié de manière directe ou indirecte, par l'intermédiaire d'un noeud, à ce point central et il ne peut être relié à un autre hôte. Tous les noeuds sont directement liés au processeur central et ils ne le sont pas entre eux. On n'accepte donc pas de lignes multipoints entre hôtes ni entre noeuds. La *Figure 3.1* montre un exemple d'un réseau à processeur central.

Le problème de conception peut être formulé de la manière suivante. Connaissant le nombre et la position des machines hôtes, il s'agit de connecter ceux-ci au processeur central dont la position est connue. Ceci peut être fait en reliant les hôtes directement au processeur ou en passant par des noeuds. Pour placer ces noeuds, un ensemble de positions possibles est disponible au départ. Il faut choisir chaque position de noeud dans cet ensemble. Le sous-ensemble résultant détermine ainsi les positions effectives des noeuds. La position des noeuds étant connue, il faut encore choisir les lignes pour connecter les hôtes aux noeuds, c'est-à-dire il faut déterminer quel hôte est connecté à quel noeud. La capacité de ces lignes dépend du trafic des hôtes. La capacité de la ligne entre un noeud et le processeur central est déterminée par le trafic des hôtes reliés à ce noeud. La solution obtenue doit bien sûr vérifier les critères de performance et minimiser le coût.

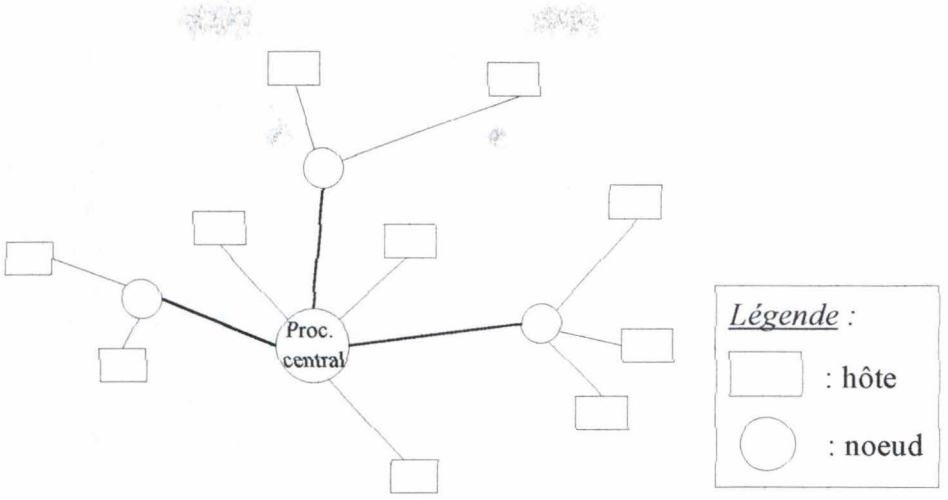

Figure 3.1 : Exemple d'un réseau à processeur central

La conception de tels réseaux a été étudiée par de nombreux auteurs dans les années septante. Le lecteur intéressé trouvera des algorithmes de Martin dans [MAR72], de Bahl et Tang dans [BAH72] et de Woo et Tang dans [WOO73]. Martin explique une méthode qui permet de connecter des noeuds à une entité centrale en utilisant cette topologie sous forme d'étoile (en anglais : star topology). Woo et Tang ont développé un algorithme semblable. Ces auteurs proposent également des solutions avec lignes multipoints. Deux algorithmes célèbres et performants sur la conception de réseaux sans lignes multipoints ont été mis au point par Bahl et Tang, appelés algorithmes d'insertion et d'élimination. La section 3.2 ci-après reprend les grandes lignes de ces deux derniers algorithmes.

De nombreuses recherches ont été nécessaires pour maîtriser ce problème de conception. Ces recherches ont fait que ce type de réseau est aujourd'hui relativement facile à concevoir à l'aide d'algorithmes comme ceux énoncés ci-dessus.

Réseau à structure distribuée

Un réseau distribué est un réseau constitué d'hôtes et de noeuds sans pour autant posséder un noeud central. Les hôtes sont toujours connectés directement à des noeuds comme dans le cas précédent; autrement dit, il n'y a pas de lignes multipoints. Les noeuds de leur part sont connectés entre eux. On divise souvent ces réseaux en deux catégories différentes : les réseaux d'accès local et le réseau de communication (aussi appelé réseau de transit). La *Figure 3.2* montre un réseau distribué et sa décomposition.

Un réseau d'accès local est un réseau où les hôtes sont connectés localement et directement à un seul noeud (central pour ce réseau). Le réseau de transit n'est constitué que de noeuds intermédiaires et de lignes entre ces noeuds. Tout le trafic non-local passe par ces noeuds et est véhiculé sur les lignes qui les relient. Il constitue en quelque sorte le "coeur" d'un réseau.

Le problème de la conception d'un réseau distribué ne peut pas être résolu par une étude exhaustive. Une des raisons est que le nombre de combinaisons possibles de connexions entre hôtes et noeuds et surtout entre noeuds croît de façon exponentielle lorsque le nombre de noeuds augmente. En plus, la complexité du problème a pour conséquence qu'il n'existe aucune méthode analytique qui permet de trouver une topologie de réseau optimale. Une solution consiste à choisir une approche heuristique, mais elle n'est guère plus facile. La solution généralement adoptée consiste à diviser le problème en une série de sous-problèmes qui sont traités séparément, soit par des méthodes analytiques, soit par des méthodes heuristiques.

Rappelons l'idée principale de la conception d'un réseau distribué. La position des machines "hôtes" ainsi que le trafic entre hôtes sont connus. Le but est de trouver le nombre de noeuds, leur emplacement en sélectionnant les noeuds parmi un ensemble de positions possibles et la position de toutes les lignes avec la capacité correspondante.

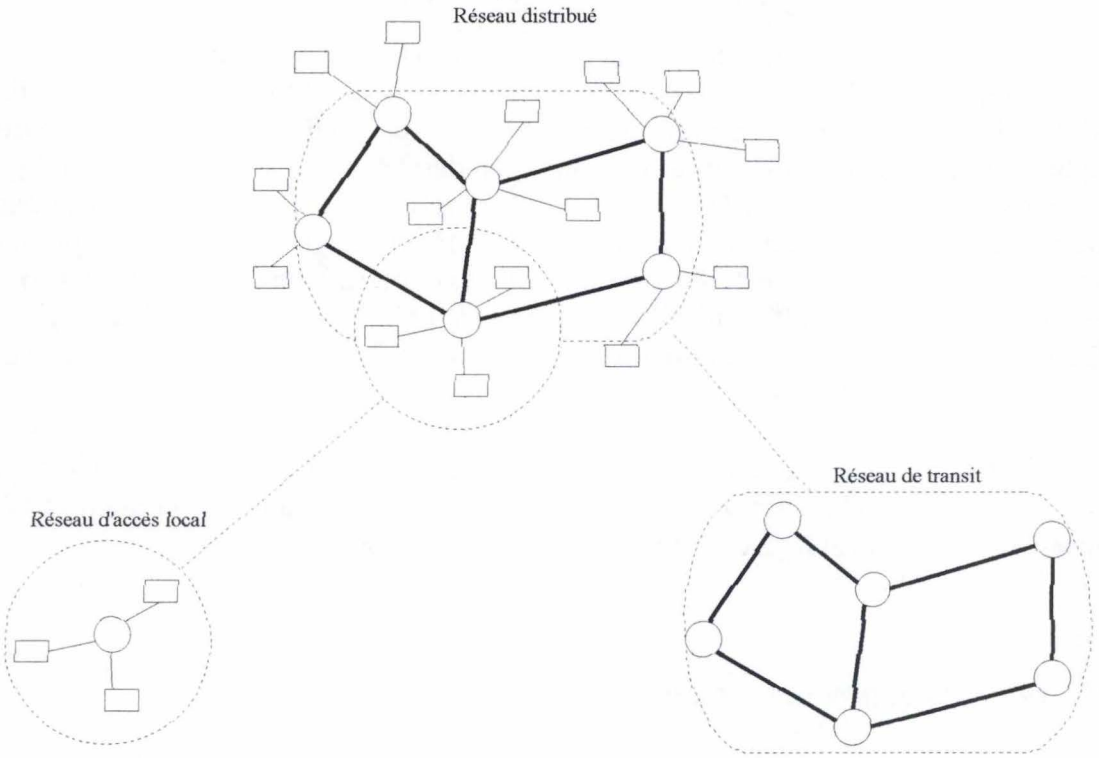

Figure 3.2 : Découpe du problème de conception d'un réseau

Il y a donc deux approches pour résoudre ce problème. Dans la première approche, on essaie de trouver une topologie optimale du réseau complet par un algorithme heuristique. Cet algorithme s'applique au réseau distribué tout entier. Etant donné la complexité de ces algorithmes, elle n'apporte pas d'avantage par rapport à une décomposition du problème. Pour cela, nous n'allons pas en dire plus.

La deuxième approche consiste à diviser le problème lors de la conception d'un réseau de communication en deux sous-problèmes (*Figure 3.2*) : d'une part la conception d'un réseau de communication et d'autre part la conception de réseaux d'accès local. Etant donné la structure des réseaux (réseaux d'accès local reliés à un réseau de transit), la décomposition permet de considérablement simplifier la

conception. En traitant le problème des réseaux d'accès local, on peut focaliser les efforts sur la recherche des réseaux de transit.

Conception de réseaux d'accès local

Dans le cas de la conception des réseaux d'accès local, il s'agit de placer les lignes de manière telle que les hôtes soient reliés aux noeuds du réseau de transit. La position des noeuds doit être déterminée avant la conception des réseaux d'accès local. Tous ces hôtes seront liés directement à un noeud, puisque les lignes multipoints ne sont pas admises. Remarquons que le réseau d'accès local est un réseau à structure centralisée bien particulier : il y a un seul noeud central et pas de noeud intermédiaire. L'algorithme qui construit des réseaux de cette catégorie est très simple : il relie chaque hôte au noeud le plus proche.

La conception des réseaux d'accès local fournit les flux de communication pour le réseau de transit associé aux réseaux locaux. Remarquons que le nombre de noeuds et leur emplacement doivent être déterminés par un autre algorithme.

Conception de réseaux de transit

La conception de réseaux d'accès local fournit les éléments nécessaires à la conception du réseau de transit. Cette dernière devra déterminer le positionnement des lignes et leur capacité de manière telle que les critères de performance (taux d'occupation par exemple) et les restrictions (p. ex. nombre limité de noeuds) sont vérifiés tout en minimisant les coûts du réseau. Dans le cadre de ce mémoire, on exigera aussi que le réseau de transit soit de connectivité d'ordre deux pour avoir un chemin de secours ("backup") lorsqu'une ligne tombe en panne. Autrement dit, il faut qu'il existe deux chemins distincts (n'ayant aucun ligne commune) entre deux noeuds quelconques.

L'algorithme de Steiglitz (cf. [STE69]) et l'algorithme dual de l'arbre minimal (cf. [PRI57]) sont des algorithmes ou heuristiques qui poursuivent cette optique. La méthode de la découpe en deux sous-réseaux ("Cut Saturation Algorithm" - cf. [GER74]) permet d'optimiser le coût total d'un réseau de transit existant.

Conception du réseau distribué composé

La conception d'un réseau distribué composé d'un réseau de transit et de réseaux d'accès local est une tâche plus complexe. Aujourd'hui, on ne trouve dans la littérature pas d'algorithme unique conçu pour ce problème bien spécifique. Nous allons tout de même donner deux propositions. Comme déjà vu, on peut

- soit combiner un algorithme de réseaux de transit avec un algorithme de réseaux d'accès local;

- soit utiliser un algorithme heuristique de type général qu'on essaiera d'adapter à la conception de réseaux composés.

Les deux approches ont des avantages et des inconvénients. L'avantage de la première est que les deux algorithmes utilisés ont été développés pour la conception de réseaux et qu'ils sont donc assez performants pour le problème simple correspondant. Malheureusement, la combinaison de l'algorithme de réseaux d'accès local avec un algorithme de réseau de transit diminue les performances - surtout en temps de calcul.

Nous allons maintenant proposer une démarche[1] à suivre lorsqu'on combine les deux types d'algorithmes :

1. Choix du nombre de noeuds;
2. *Recherche de l'emplacement des noeuds;*
3. Recherche des lignes entre hôtes et noeuds par un algorithme de conception d'un réseau d'accès local. Cet algorithme détermine d'abord à quel noeud chaque hôte est connecté;
4. Connexion des noeuds entre eux en minimisant le coût global du réseau par un algorithme de conception d'un réseau de transit;
5. Répétition des étapes 1-4 en modifiant le nombre de noeuds jusqu'à ce qu'on trouve une solution satisfaisante.

Chaque étape de cette démarche peut être accomplie par différentes méthodes ou algorithmes. On peut par exemple au point 1 choisir le nombre de noeuds par hasard. L'emplacement (point 2) peut être trouvé par hasard ou par l'algorithme de placement de noeuds expliqué dans la section 3.3. Le point trois, c'est-à-dire la recherche des lignes entre hôtes et noeuds, est généralement atteint par l'algorithme de distance minimale. Les lignes entre noeuds peuvent par exemple être choisis par hasard. Une approche parmi d'autres est d'introduire un algorithme comme l'algorithme de Steiglitz ou l'algorithme dual de l'arbre minimal (lequel est utilisé dans le programme accompagnant ce mémoire). La dernière étape est la plus importante : elle permet de trouver une solution satisfaisante. C'est l'algorithme principal qui doit diriger la recherche de la solution optimale. Au mieux cet algorithme est adapté au problème, au plus vite on converge vers un optimum. Cet algorithme doit donc utiliser les méthodes citées ci-dessus pour trouver un réseau optimal. L'algorithme du recuit simulé (cf. [AAR89]) qui fait l'objet de ce mémoire est un de ceux-ci.

Le plus grand inconvénient de la deuxième approche est que c'est un algorithme général qui n'est pas "taillé sur mesure". Ce type d'algorithme est applicable à de nombreux problèmes complexes, mais sans pouvoir répondre aux caractéristiques du problème posé. En plus, la complexité et le nombre des calculs liés à celle-ci sont tels que le regroupement des calculs en un seul algorithme rendent la tâche de conception très difficile. Le nombre de combinaisons de modifications possibles rendent une convergence rapide impossible. Les performances d'un tel algorithme seront donc probablement très faibles. Malheureusement, ces algorithmes ne garantissent pas, comme d'ailleurs les autres algorithmes du problème, de trouver une solution optimale globale.

[1] *Une démarche analogue a été proposée par R.L. Sharma dans [SHA90]*

L'objectif de ce chapitre est de donner une brève explication des algorithmes mentionnés ci-dessus. A la fin du chapitre, on essayera de comparer les algorithmes. Dans le chapitre suivant, on expliquera plus en détail l'algorithme utilisé dans le programme qui accompagne ce mémoire.

3.2. Réseaux à structure centralisée

Les algorithmes d'insertion et d'élimination (cf. [BAH72]) ont été conçus pour les réseaux centralisés, c'est-à-dire des réseaux à un processeur central. Même si le but de ce mémoire est la conception des réseaux distribués, il nous semble intéressant d'expliquer brièvement les réseaux à structure centralisée.

Tous les terminaux d'un ordinateur puissant (de type mainframe) doivent être connectés à un processeur central. Dans ces algorithmes, on groupe souvent plusieurs terminaux dans des concentrateurs, pour ne pas connecter les terminaux un à un au processeur central. Les concentrateurs de leur part sont reliés à l'unité centrale. Tandis que les terminaux restent toujours au même endroit, les concentrateurs peuvent être mis à des endroits stratégiques. Le but de l'algorithme est de trouver les endroits des concentrateurs en minimisant les coûts de cablage et les coûts des concentrateurs.

Dans le cas des réseaux à grande distance ayant une structure centralisée, le principe est à peu près le même. On vise à optimiser la connexion des hôtes à un noeud central : les machines hôtes peuvent être connectées à un noeud intermédiaire ou directement liées au noeud central, tandis que les noeuds intermédiaires de leur part doivent être connectés directement au noeud central (*Figure 3.3*). Les algorithmes d'insertion et d'élimination de Bahl et Tang s'adaptent donc particulièrement bien aux types de réseaux qu'on rencontre ici.

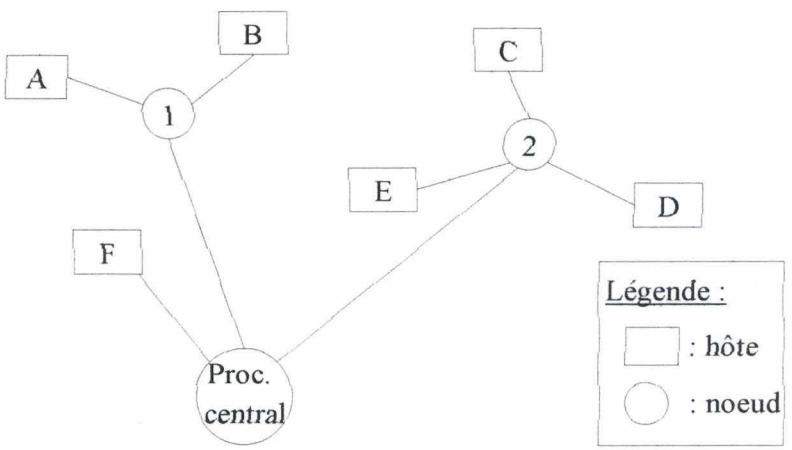

Figure 3.3 : Exemple d'un réseau à processeur central

Dans les réseaux à structure centralisée, les positions des hôtes sont supposées connues et les noeuds peuvent être placés parmi un ensemble de positions possibles (*Figure 3.4*). Les algorithmes doivent alors trouver un sous-ensemble de noeuds auxquels les hôtes seront connectés. Examinons maintenant les algorithmes plus en détail.

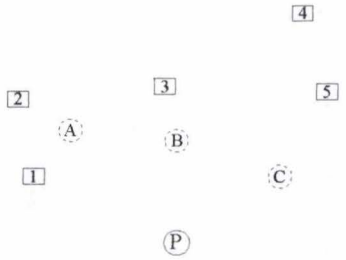

Figure 3.4 : Problème de départ des algorithmes d'insertion et d'élimination

Algorithme d'insertion

Le principe de l'algorithme d'insertion est très simple : au début, tous les hôtes sont directement connectés au noeud central. Lors des itérations de l'algorithme, on ajoute au fur et à mesure des noeuds vers lesquels on relie les hôtes. Un noeud est seulement ajouté si son introduction entraîne une diminution du coût total du réseau. La *Figure 3.5* montre l'organigramme de cet algorithme.

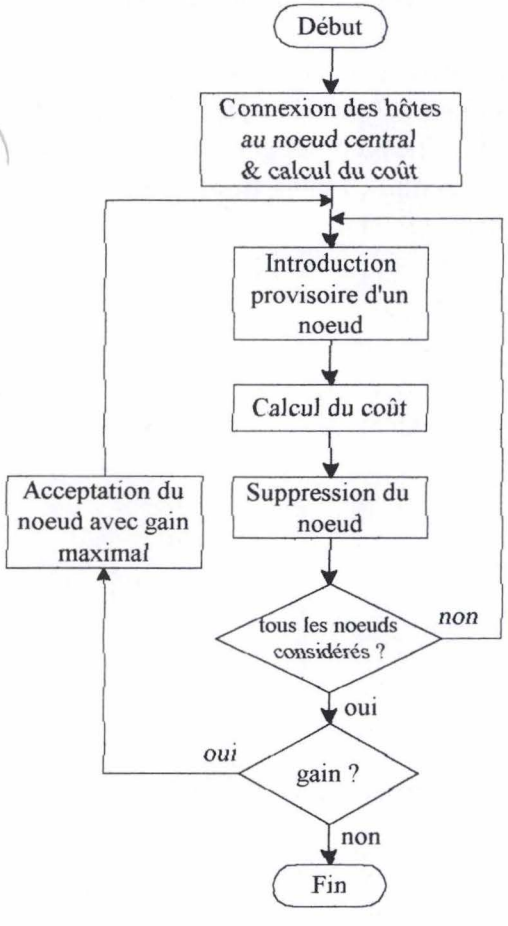

Figure 3.5 : Algorithme d'insertion

Au départ, la position des hôtes et du processeur central sont connus. En plus, il y a un ensemble de positions potentielles sur lesquels on peut placer les noeuds. La *Figure 3.4* représente cette situation : 1, 2, ..., 5 représentent les positions des hôtes et A, B, C sont des endroits où on peut placer des noeuds; P représente le processeur central. On connaît également les coûts de connexion des hôtes vers le noeud central et les coûts de connexion de chaque hôte vers toutes les positions possibles.

Le *Tableau 3.1* montre les coûts pour l'exemple de la *Figure 3.4*.

Noeuds \ Hôtes	1	2	3	4	5
A	1	1	2	5	5
B	3	3	1	3	3
C	5	5	3	3	2
P	3	4	3	5	4

Tableau 3.1 : Matrice des coûts de lignes

En plus on sait calculer les coûts entre les positions potentielles et le noeud central; ils sont exprimés dans le *Tableau 3.2*. Remarquons qu'on ne donne pas les unités dans les exemples de ce chapitre pour ne pas compliquer les calculs; on parlera tout simplement d'unités en général.

Noeuds :	A	B	C
P	3	2	2

Tableau 3.2 : Matrice des coûts

La première étape de l'algorithme d'insertion consiste à relier directement tous les hôtes au noeud central. Le noeud central est donc le seul noeud du réseau. Ensuite, le coût de cette topologie est évalué à partir de la matrice des coûts de ligne entre hôtes et processeur central. Dans le cas de l'exemple ci-dessus, ce coût vaut :

$$\begin{aligned} C_{total} &= C_{1\text{-}P} + C_{2\text{-}P} + C_{3\text{-}P} + C_{4\text{-}P} + C_{5\text{-}P} \\ &= 3 + 4 + 3 + 5 + 4 = 19 \end{aligned}$$

L'objectif de l'étape suivante est de réduire le coût du réseau en introduisant un noeud intermédiaire. On pose un noeud à un endroit de placement potentiel et on y relie les hôtes qui diminuent le coût global du réseau. Le coût total du réseau lors de l'ajout du noeud A de la *Figure 3.4* peut être trouvé de la manière suivante :

$$\begin{aligned} C_{total} &= C_{A\text{-}P} + \sum_{i=1} \min(C_{i\text{-}A}, C_{i\text{-}P}) \\ &= C_{A\text{-}P} + C_{1\text{-}A} + C_{2\text{-}A} + C_{3\text{-}A} + C_{4\text{-}A} + C_{5\text{-}P} \\ &= 3 + 1 + 1 + 2 + 5 + 4 = 16 \end{aligned}$$

De la même manière, on peut calculer le coût global du réseau pour les deux autres noeuds. Lorsqu'on ajoute le noeud B, le coût sera :

$$C_{total} = C_{B\text{-}P} + C_{1\text{-}P} + C_{2\text{-}B} + C_{3\text{-}B} + C_{4\text{-}B} + C_{5\text{-}B}$$
$$= 2 + 3 + 3 + 1 + 3 + 3 = 15$$

et le coût global avec noeud C est de 17 unités.

Le noeud qui permet de réduire le plus le coût du réseau entier par rapport au coût initial est alors retenu. Dans le cas de notre exemple, l'ajout du noeud B diminue le coût de 4 unités. L'algorithme d'insertion (*Figure 3.5*) va donc ajouter ce noeud.

Cette procédure est répétée en essayant à chaque étape d'introduire de la même manière un noeud supplémentaire dans le réseau tant que le coût total du réseau décroît. Remarquons qu'à chaque étape, tous les hôtes sont connectés au nouveau noeud même s'il a été relié à un autre noeud. Donc, même si un hôte est connecté à un noeud intermédiaire, le coût total du réseau peut diminuer lorsqu'on le retire et si on le relie à un autre noeud. L'algorithme s'arrête lorsqu'il n'est plus possible de réduire le coût total du réseau.

Ainsi, l'itération suivante de notre exemple donne les résultats suivants : En introduisant le noeud A, le coût global correspondant est :

$$C_{total} = C_{A\text{-}P} + C_{B\text{-}P} + C_{1\text{-}A} + C_{2\text{-}A} + C_{3\text{-}B} + C_{4\text{-}B} + C_{5\text{-}B}$$
$$= 3 + 2 + 1 + 1 + 1 + 3 + 3 = 14$$

et lorsqu'on ajoute C, le coût vaut :

$$C_{total} = 2 + 2 + 3 + 3 + 1 + 3 + 2 = 16$$

L'ajout du noeud A - en plus du noeud B - permet donc de réduire encore une fois le coût global du réseau. On voit que l'hôte 2, qui était à l'initialisation connecté au processeur central et à l'itération suivante au noeud B, est finalement relié au noeud A.

La troisième itération essaie d'ajouter le noeud C, mais cette opération ne permet plus de réduire le coût, puisque le coût résultant est :

$$C_{total} = 2 + 3 + 2 + 1 + 1 + 1 + 3 + 2 = 15.$$

Comme on vient de le voir, cet algorithme permet d'introduire les deux noeuds A et B afin de réduire le coût de 19 unités au départ, à 14 unités après les trois itérations. Le réseau de la *Figure 3.4* aura, après introduction des noeuds, la topologie représentée à la *Figure 3.6* avec un coût de 14 unités.

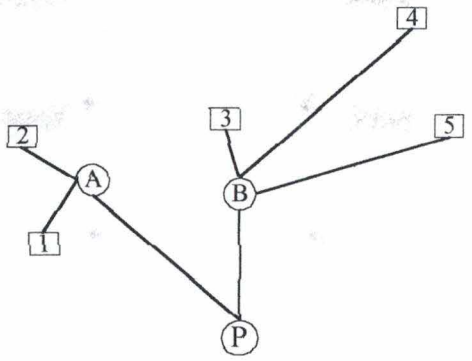

Figure 3.6 : Réseau après l'application de l'algorithme d'insertion

Algorithme d'élimination

Cet algorithme fait, comme son nom l'indique, le contraire de l'algorithme précédent : il retire des noeuds au fur et à mesure. L'étape initiale consiste à placer tous les noeuds intermédiaires dans le réseau et à relier les hôtes au noeud le plus proche. Une façon de relier les hôtes au noeud le plus proche est expliquée dans la section 3.3. Cette notion de proximité s'exprime soit en termes de distance, soit en termes de coûts de connexion. Autrement dit, on installe pour chaque hôte les lignes pour lesquelles le coût ou la distance est minimal.

Cette situation est représentée à la *Figure 3.7*. Remarquons que les hôtes, les noeuds et le processeur sont les mêmes que ceux de l'exemple précédent (l'exemple de l'algorithme d'insertion). Nous allons donc considérer les coûts du *Tableau 3.1* et du *Tableau 3.2*.

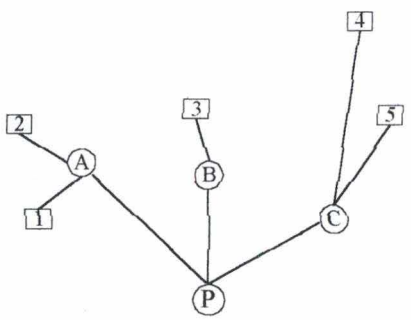

Figure 3.7 : Réseau dont les hôtes sont reliés aux noeuds les plus proches

Le coût total du réseau ci-dessus - en tenant compte des coûts des tableaux *Tableau 3.1* et *Tableau 3.2* - se calcule de la manière suivante :

$$C_{total} = C_{A\text{-}P} + C_{B\text{-}P} + C_{C\text{-}P} + C_{1\text{-}A} + C_{2\text{-}A} + C_{3\text{-}B} + C_{4\text{-}C} + C_{5\text{-}C}$$
$$= 3 + 2 + 2 + 1 + 1 + 1 + 3 + 2 = 15$$

La suite de l'algorithme vise à retirer un noeud à la fois (par itération) afin de réduire le coût du réseau. Les hôtes ainsi déconnectés sont reliés à d'autres noeuds intermédiaires ou au noeud central de façon telle que le coût soit minimal.

Lorsqu'on enlève le noeud A de notre exemple, on obtient les coûts suivants :

$$C_{total} = C_{B\text{-}P} + C_{C\text{-}P} + C_{1\text{-}B} + C_{2\text{-}B} + C_{3\text{-}B} + C_{4\text{-}C} + C_{5\text{-}C}$$
$$= 2 + 2 + 3 + 3 + 1 + 3 + 2 = 16$$

L'élimination du noeud B entraîne une réduction du coût à 14 unités. Le retrait du noeud C fait également réduire le coût à 14 unités.

Le noeud qui entraîne la réduction du coût la plus grande est finalement retiré. Dans notre cas, le retrait des noeuds B et C génère un coût minimal. Comme B est le premier, l'algorithme va le retirer après la première itération. Cette étape est répétée tant que le coût diminue lors d'une élimination d'un noeud (*Figure 3.8*). L'algorithme s'arrête lorsqu'il n'est plus possible de réduire le coût en éliminant un noeud.

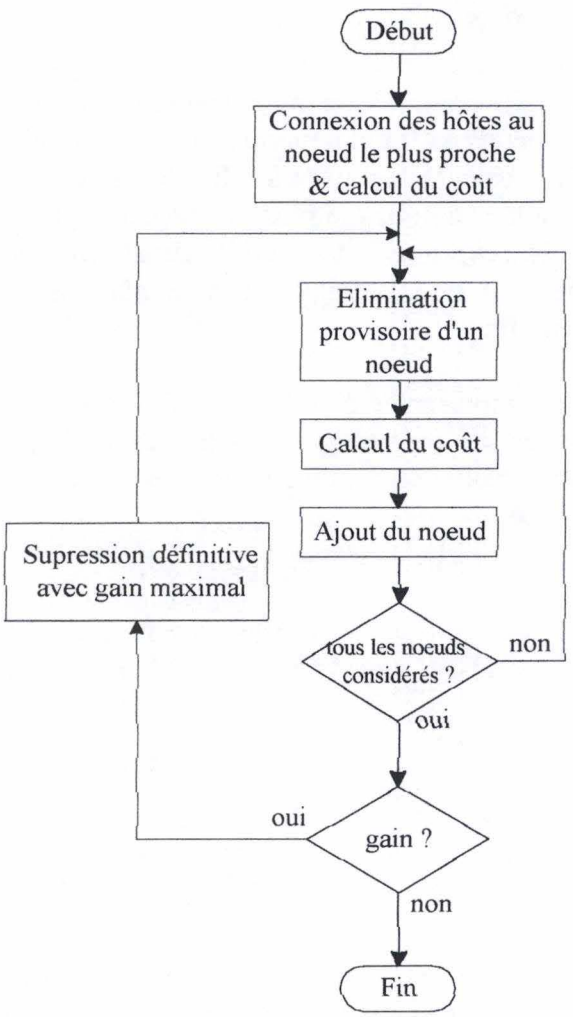

Figure 3.8 : Algorithme d'élimination

En ce qui concerne notre exemple, le retrait d'un deuxième noeud - après le noeud B - ne permet plus de diminuer le coût : en retirant le noeud A, le coût total du réseau sera:

$$C_{total} = C_{C\text{-}P} + C_{1\text{-}P} + C_{2\text{-}P} + C_{3\text{-}P} + C_{4\text{-}C} + C_{5\text{-}C}$$
$$= 2 + 3 + 4 + 3 + 3 + 2 = 17$$

En éliminant C, le coût est de 16 unités. L'algorithme d'élimination s'arrête donc ici. Le réseau résultant est montré à la *Figure 3.9*.

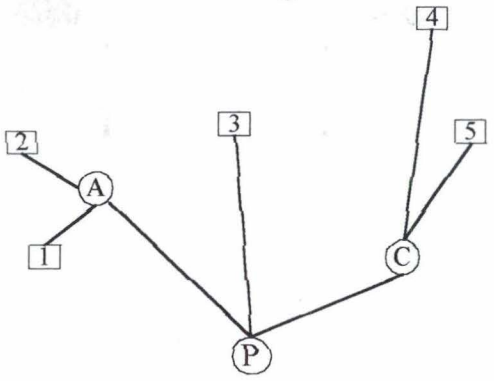

Figure 3.9 : Réseau après l'application de l'algorithme d'élimination

Bahl et Tang ont trouvé que si les réseaux sont complexes, les résultats de ces deux algorithmes ne sont pas forcément les mêmes puisqu'ils ne garantissent pas d'arriver à une solution d'optimum global. Notre exemple en témoigne : l'algorithme d'insertion trouve une solution ayant un coût minimal de 13 unités et l'algorithme d'élimination un coût de 14 unités en partant des mêmes éléments de base.

3.3. Réseaux d'accès local

Le but de l'étape de conception de réseaux d'accès local consiste à trouver tous les réseaux d'accès local d'un réseau distribué. Or, un réseau d'accès local est formé à partir d'un noeud central et des hôtes qui entourent ce noeud. Le noeud est le centre local des hôtes qui l'entourent. Avant de relier les hôtes à leur noeud local, la position des noeuds doit être fixe; il faut donc connaître l'emplacement de tous les noeuds. Cet emplacement peut être trouvé par un positionnement "par hasard" ou par un algorithme.

Sharma a trouvé une méthode simple pour déterminer les positions où on peut placer les noeuds (cf. [SHA90]). Cette méthode se base sur la recherche de centres de gravité et est expliquée ci-dessous dans l'algorithme de placement des noeuds.

Ayant déterminé les positions des noeuds par cet algorithme, on peut passer à la construction des réseaux d'accès local à l'aide d'un algorithme très simple (cf. "algorithme des réseaux d'accès local"). Mais nous allons d'abord présenter la méthode de la recherche des centres de gravité.

Algorithme de placement de noeuds

Le problème de placement de noeuds consiste à localiser un nombre donné de noeuds parmi un ensemble donné de positions possibles. Le nombre de noeuds doit être connu avant le lancement de l'algorithme. L'algorithme développé par Sharma (cf. [SHA90]) est fondé sur le concept du centre de gravité et d'une décomposition de la topologie en différentes régions :

On cherche d'abord le centre de gravité de l'ensemble des positions possibles connues au départ. Les coordonnées X_{CG} et Y_{CG} de ce centre peuvent être trouvé par la formule suivante :

$$\begin{cases} X_{CG} = \left[\sum_i X_i \cdot TP_i\right] \Big/ \left[\sum_i TP_i\right] \\ Y_{CG} = \left[\sum_i Y_i \cdot TP_i\right] \Big/ \left[\sum_i TP_i\right] \end{cases} \qquad (3.1)$$

où Xi et Yi sont respectivement les coordonnées horizontales (longitude) et verticales (latitude) de la $i^{ème}$ position (où i = 1,..., NbNoeuds)
TPi est le trafic moyen pour la position i

Cette formule montre que le centre de gravité est pondéré par le trafic du réseau. Il représente donc le centre de la répartition du trafic moyen du réseau. Nous appelons trafic moyen pour la position i, la somme des trafics moyens émis et reçus par tous les hôtes voisins à cette position. On dit qu'un hôte est voisin d'une position, lorsque cette position est celle qui est la plus proche de cet hôte. La proximité d'une position peut se mesurer soit :

- par la distance à vol d'oiseau
- par la distance au sens Belgacom (distance zonale + interzonale + zonale)
- par le coût Belgacom (cf. Chapitre 2)

Dans les exemples de ce chapitre, nous allons utiliser les distances (coûts) indiquées dans les tableaux ou sur les figures correspondantes. Dans l'algorithme implémenté dans le programme accompagnant ce mémoire, on prend la distance au sens Belgacom. Remarquons que l'algorithme d'accès local expliqué ci-dessous trouve les hôtes voisins d'un noeud.

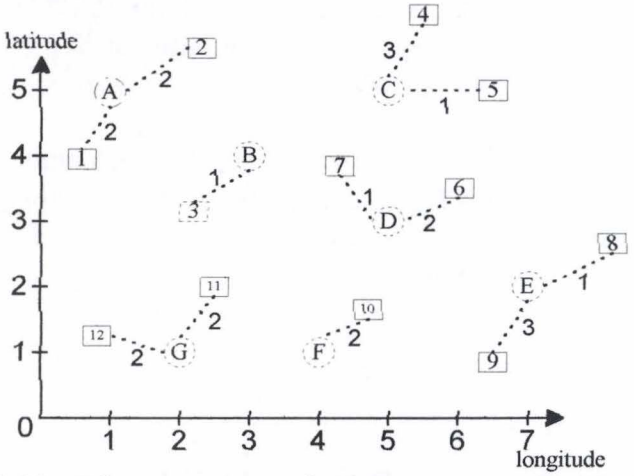

Figure 3.10 : Réseau montrant les hôtes et positions potentielles

La *Figure 3.10* montre une topologie composée de positions potentielles (A, B, ...) et d'hôtes (1, 2, ...). Les lignes pointillées entre hôtes et positions sur cette figure indiquent que l'hôte est voisin de la position. A côté de cette ligne, on trouve le trafic moyen que cet hôte émet et reçoit de la position. La *Figure 3.11* montre la topologie des positions avec leur trafic moyen résultant de la somme des trafics moyens de leurs hôtes voisins. Elle indique également ses coordonnées dans le système cartésien, ce qui est important pour le calcul des distances.

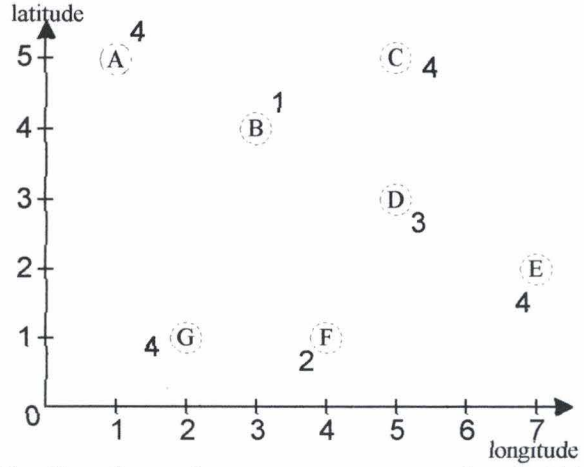

Figure 3.11 : Topologie des positions avec coûts des hôtes voisins

Revenons au calcul du centre de gravité. L'algorithme va d'abord chercher le centre de gravité de tous les noeuds potentiels, puis diviser la topologie totale en deux régions. Après, il calcule un centre de gravité pour chaque région et divise de nouveau en régions jusqu'à ce qu'on obtienne autant de centres de gravités qu'il faut. A titre d'exemple, nous allons chercher quatre endroits pour placer des noeuds parmi les six positions possibles. Dans l'exemple de la *Figure 3.10*, la formule (3.1) permet de calculer le centre de gravité suivant :

$$\begin{cases} X_{CG} = \frac{4+3+20+15+28+8+8}{4+1+4+3+4+2+4} = \frac{86}{22} = 3{,}9 \\ Y_{CG} = \frac{20+4+20+9+8+2+4}{4+1+4+3+4+2+4} = \frac{67}{22} = 3{,}1 \end{cases}$$

Si on ne cherche qu'un seul centre de gravité pour tout le réseau, il suffit alors de trouver la position la plus proche et d'y placer le noeud.

Pour trouver deux nouvelles positions de noeuds, on "trace" une ligne verticale à travers le centre de gravité trouvé ci-dessus. Remarquons qu'on pourrait aussi tracer une ligne horizontale passant par le centre. Ce choix dépend de la répartition du trafic dans le réseau. On forme ensuite deux ensembles - un ensemble des positions à gauche de la ligne et l'autre des positions à droite. On calcule alors le centre de gravité de ces deux régions par la même formule (3.1) décrite ci-dessus. Sur la *Figure 3.13*, qui montre les centres de gravité en général, ce seront les positions CG2 et CG3. De nouveau, on cherche les positions les plus proches des deux centres de gravités et de cette manière, on trouve deux nouveaux endroits pour placer des noeuds.

Si on veut avoir trois noeuds, il suffit de prendre le centre de gravité de la région complète et les centres des deux sous-régions.

S'il faut quatre noeuds, on trace une ligne horizontale par le centre de gravité et on calcule le centre de gravité pour les quatre régions. Comme on cherche quatre noeuds dans notre exemple, on doit donc également décomposer le réseau (la topologie) en quatre régions en traçant une ligne verticale et horizontale passant par le centre de gravité trouvé ci-dessus. On obtient les régions W, X, Y et Z (*Figure 3.12*). Comme il n'y a qu'une seule position dans les régions X et Y, ce sont en même temps leurs centres. Calculons donc d'abord le centre de gravité de la région W. Les positions potentielles y sont A et B. Leur centre de gravité pondéré par le trafic moyen donne comme résultat :

$$\begin{cases} X_{CG} = \frac{4+38}{4+1} = \frac{7}{5} = 1{,}4 \\ Y_{CG} = \frac{20+4}{4+1} = \frac{24}{5} = 4{,}8 \end{cases}$$

La position A est la position la plus proche de ce point (*Figure 3.12*).

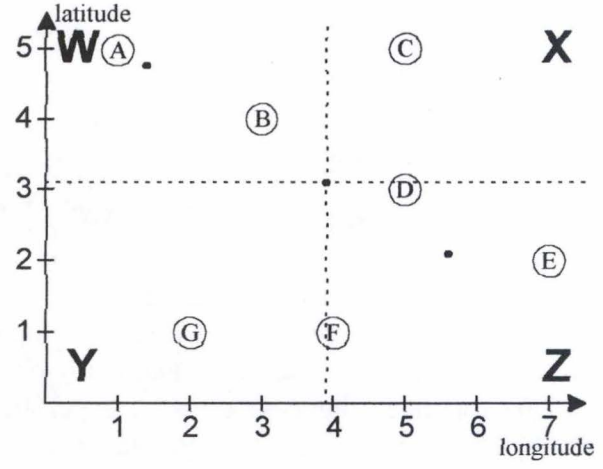

Figure 3.12 : Centres de gravité et noeuds placés

De même le centre de gravité de la région Z se calcule en appliquant la formule (3.1) aux trois positions D, E et F. Dans cette région, la position G est la plus proche du centre de gravité (5,7 ; 2,1). On cherche les positions les plus proches et ainsi on trouve quatre nouveaux emplacements de noeuds. Les quatre positions dans notre exemple sont donc A, C, D et G (*Figure 3.12*).

S'il faut encore plus de noeuds, on répète cette décomposition en sous-régions par des lignes verticales et horizontales jusqu'à ce qu'on obtienne assez de noeuds. L'alternance entre lignes horizontales et verticales est nécessaire pour garantir une répartition équilibrée.

La *Figure 3.13* illustre le principe de la découpe en régions et montre les centres de gravité correspondants. Comme on le voit sur cette figure, on a la possibilité de choisir entre un et sept positions de noeuds. Plus précisément, lorsqu'on veut placer un noeud il suffit de choisir CG1. S'il faut deux noeuds, on prendra CG2 et CG3; pour avoir trois noeuds, on peut prendre CG1, CG2 et CG3. Si on est à la recherche de quatre noeuds, on prendra CG4, CG5, CG6 et CG7, etc.

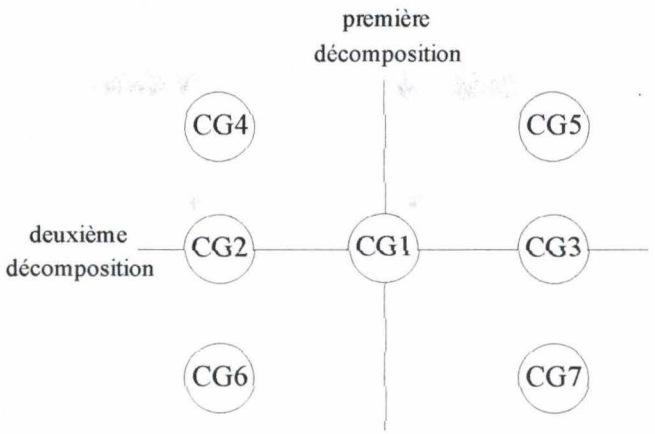

Figure 3.13 : Décomposition en régions et centres de gravité

Pour améliorer le résultat, on peut tenir compte de la répartition du trafic dès la première décomposition. Si la répartition du trafic est plutôt orientée nord-sud, il faut commencer la décomposition du réseau par une ligne horizontale. Si l'orientation est ouest-est, il est préférable de commencer par une ligne verticale.

Remarquons que Sharma signale que cet algorithme ne trouve pas toujours la répartition idéale des noeuds. Mais, malgré sa simplicité, les résultats sont très semblables à ceux d'algorithmes très sophistiqués qui ont été construits explicitement pour ce problème. Comme il faut en plus combiner cet algorithme avec deux autres, la simplicité constitue un avantage considérable.

Algorithme des réseaux d'accès local

Le nombre de noeuds et leur emplacement étant connus, il faut maintenant construire les réseaux d'accès local autour de chaque noeud. Pour cela, il suffit de connecter les hôtes aux noeuds de manière telle que le coût total des réseaux d'accès local soit minimal. Or, le coût d'une ligne change en fonction de la distance entre ces extrémités (cf. chapitre 2) et avec la capacité de la ligne. Mais, comme il s'agit des lignes hôte-noeud, leur capacité dépend du trafic arrivant à l'hôte et partant de l'hôte. Ce trafic étant indépendant du noeud auquel l'hôte est connecté, la capacité de la ligne correspondante sera donc la même pour n'importe quel noeud. Par conséquent, le coût d'une telle ligne ne dépend que de sa longueur et sera minimal lorsque la longueur est minimale; ce qui veut dire qu'il suffit de relier les hôtes aux noeuds les plus proches.

Pour chaque hôte, on calcule la distance à tous les noeuds. Lorsque la distance est minimale, on relie l'hôte au noeud correspondant. On trouve ainsi le réseau d'accès local pour chaque noeud.

En résumé, on peut dire que la conception de réseau d'accès local se décompose en trois phases :
- détermination du nombre de noeuds;

- recherche de l'emplacement des noeuds (par exemple par la méthode des centres de gravité);
- construction des réseaux d'accès local en reliant les hôtes aux noeuds les plus proches.

Comme exemple de cet algorithme, on peut relier chaque hôte au noeud le plus proche de la *Figure 3.4* de la section précédente. Supposons que la distance entre ces endroits soit égale au coût. La matrice des distances entre hôtes et noeuds sera celle du *Tableau 3.1.*

Essayons d'abord de connecter l'hôte 1 a son noeud le plus proche : en tenant compte de la matrice des distances, on peut voir que le noeud A est le plus proche à l'hôte 1 et à l'hôte 2. De même, le noeud B est le noeud le plus proche à l'hôte 3. Pour montrer les distances minimales, on les a entourés dans le tableau suivant :

Noeuds \ Hôtes	1	2	3	4	5
A	(1)	(1)	2	5	5
B	3	3	(1)	(3)	3
C	5	5	3	(3)	(2)
P	3	4	3	5	4

Tableau 3.3 : Distance entre hôtes et noeuds

Ainsi, l'algorithme des réseaux d'accès local permet également de construire la situation initiale de l'algorithme d'élimination. Le résultat est représenté à la *Figure 3.7.*

Examinons maintenant les différentes possibilités (algorithmes) de construire les réseaux de transit.

3.4. Réseau de transit

Les réseaux de transit sont des réseaux qui possèdent les propriétés suivantes :

- *Critère de fiabilité :* la connectivité entre deux noeuds i et j est supérieure à une valeur fixée r_{ij} (souvent ≥ 2), c'est-à-dire il y a au moins r_{ij} chemins entre les noeuds i et j n'ayant aucune ligne en commun.
- *Critère d'optimalité :* un réseau qui satisfait à ce critère est tel qu'aucun autre réseau vérifiant le critère de fiabilité n'a des coûts inférieurs.

On appelle solution réalisable une solution qui vérifie le premier critère. Dans les exemples de ce mémoire et dans le programme, on impose au critère de fiabilité la connectivité d'ordre deux. Une solution optimale est une solution réalisable qui satisfait au critère d'optimalité. Le but de ces algorithmes est de trouver des solutions réalisables dont le coût est le plus proche possible du coût optimal.

Algorithme de Steiglitz

L'algorithme de Steiglitz (cf. [STE69]) est un algorithme qui vise à construire un réseau de transit vérifiant le critère de fiabilité et ayant un coût le plus proche possible de l'optimum global.

La méthode comprend deux parties principales (*Figure 3.14*) : la procédure de départ (starting routine) et la procédure d'optimisation (optimizing routine). La procédure de départ génère une solution réalisable. La procédure d'optimisation effectue des transformations locales afin d'obtenir un réseau de coût inférieur.

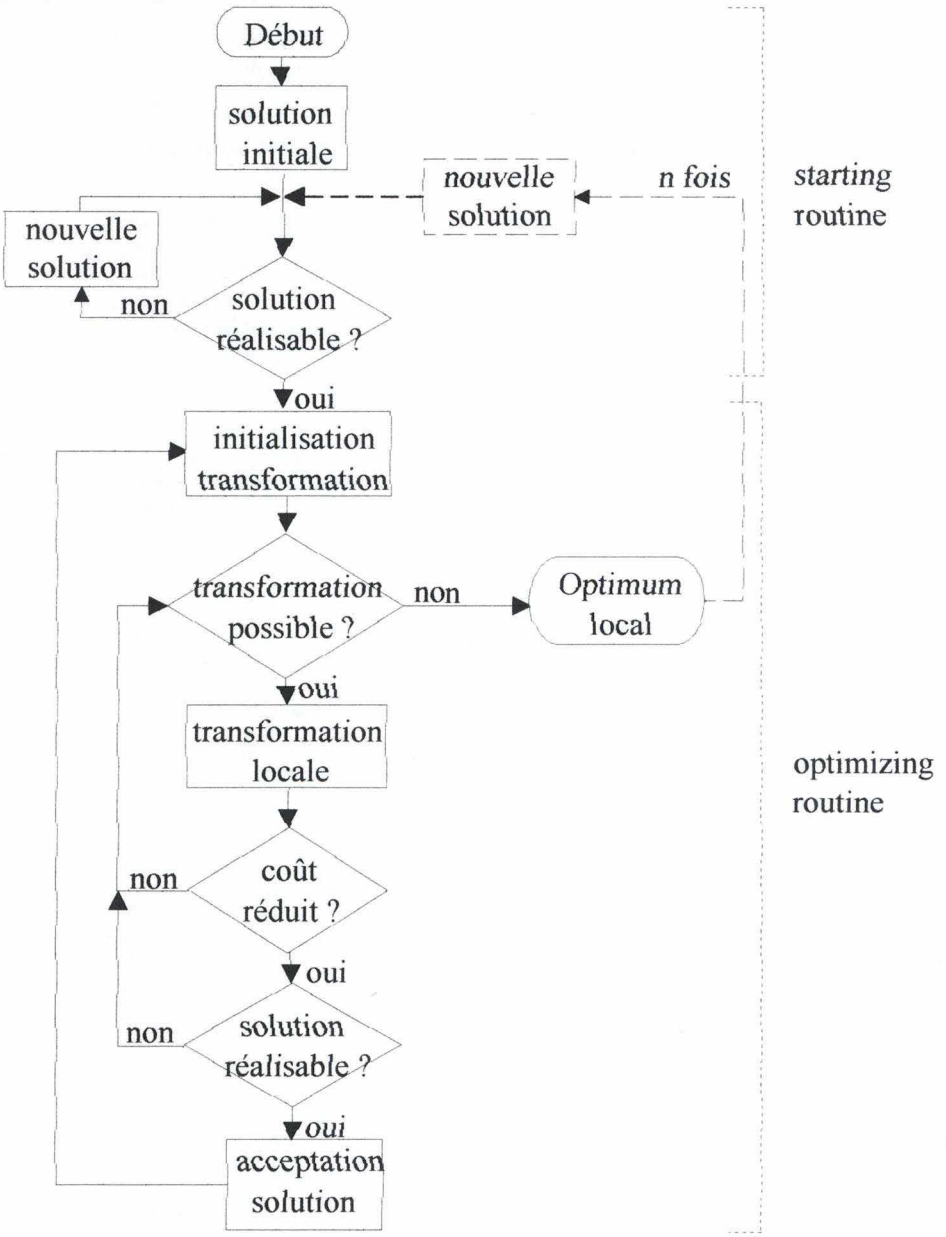

Figure 3.14 : Algorithme de Steiglitz

Il y a différentes transformations possibles. Une transformation qui est souvent utilisée est le "X-change". Elle consiste à échanger les lignes entre deux paires de noeuds (*Figure 3.15*).

Figure 3.15 : Le X-change

Lors de la procédure de départ, on cherche une solution de façon aléatoire et on vérifie alors si cette solution est réalisable. Examinons le fonctionnement de l'algorithme de Steiglitz à l'aide d'un exemple de quatre noeuds. Rappelons que le critère de fiabilité impose une connectivité d'ordre deux, c'est-à-dire qu'il existe entre deux noeuds au moins deux chemins n'ayant aucune ligne en commun.

Le coût entre ces quatre noeuds est donné par la matrice suivante :

Noeuds / Noeuds	A	B	C	D
A	0	4	4	5
B	4	0	5	3
C	4	5	0	3
D	5	3	3	0

Tableau 3.4 : Matrice des coûts entre noeuds

Supposons que la génération aléatoire de la solution initiale ait donné le réseau de la *Figure 3.16*. Le coût total de ce réseau se compose de la somme des coûts des lignes de ce réseau (on ne tient pas compte des coûts des réseaux d'accès local) et est donc de 20 unités dans notre exemple. Remarquons que cette solution a une connectivité d'ordre deux et est donc réalisable. Si la solution initiale n'est pas réalisable, on génère une nouvelle.

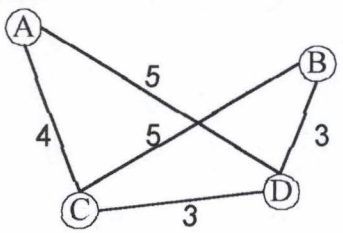

Figure 3.16 : Solution initiale

Après avoir trouvé une solution initiale réalisable, on entre dans la procédure d'optimisation. La première étape de cette procédure consiste à définir et puis à énumérer toutes les transformations possibles. Dans notre exemple, nous allons seulement utiliser la transformation du "X-change". Comme il y a quatre noeuds, on peut former trois couples de lignes entre ces quatre noeuds. Ce sont les couples

(AB,CD), (AC,BD) et (AD,BC). Chaque couple peut être transformé par le "X-change" de deux manières différentes; ce qui nous donne les six transformations possibles :

$$\begin{aligned}
(AB,CD) &\rightarrow (AC,BD) \quad (1)\\
&\rightarrow (AD,BC) \quad (2)\\
(AC,BD) &\rightarrow (AD,BC) \quad (3)\\
&\rightarrow (AB,CD) \quad (4)\\
(AD,BC) &\rightarrow (AB,CD) \quad (5)\\
&\rightarrow (AC,BD) \quad (6)
\end{aligned}$$

On applique alors une transformation au réseau et on calcule le coût. Lorsqu'une telle transformation donne lieu à un réseau réalisable et moins cher, on accepte la transformation et la recherche continue sur ce réseau, c'est-à-dire on cherche de nouvelles transformations à partir de ce réseau. Dans le cas contraire, on l'ignore et on applique une nouvelle modification.

Dans notre exemple, les deux premières transformations visent à modifier les lignes AB et CD. Or, comme la ligne AB n'existe pas dans la solution initiale, on peut ignorer les transformations (1) et (2) pour la première itération. Appliquons donc la transformation (3); autrement dit, on remplace la ligne AC par AD et la ligne BD par BC. Malheureusement, la nouvelle solution n'est pas réalisable, car les noeuds A et B n'ont qu'une seule ligne et ne vérifient pas le critère de fiabilité. Essayons donc la transformation (4). La ligne AC devient AB et la ligne BD devient CD. Le réseau résultant est réalisable et est représenté sur la *Figure 3.17.*

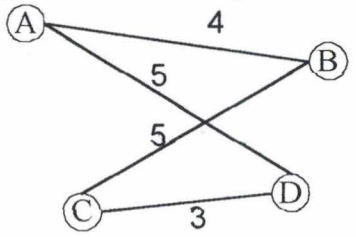

Figure 3.17 : Solution après la première itération

Le coût de cette solution vaut :

$$\begin{aligned}
C_{total} &= C_{AB} + C_{BC} + C_{CD} + C_{AD}\\
&= 4 + 5 + 5 + 3 = 17
\end{aligned}$$

et est inférieur au coût de la solution initiale. L'algorithme de Steiglitz va donc accepter cette solution.

A l'itération suivante, on applique de nouveau une transformation afin de trouver un coût inférieur. Cette étape est répétée jusqu'à ce qu'aucune amélioration ne soit plus possible. La solution finale est un optimum local.

Après avoir accepté la solution de la *Figure 3.17,* l'algorithme peut appliquer toutes les transformations possibles à cette nouvelle solution. En effectuant la première transformation, la ligne AB sera remplacée par la ligne AC et CD par BD ce qui n'est

pas une solution réalisable. L'application de (2) donne le réseau de transit de la *Figure 3.18* lequel est réalisable. Son coût se calcule par la somme des coûts de lignes :

$$C_{total} = C_{AB} + C_{BD} + C_{CD} + C_{AC}$$
$$= 4 + 4 + 3 + 3 = 14$$

Comme il est inférieur à celui de l'itération suivante, la nouvelle solution sera également acceptée.

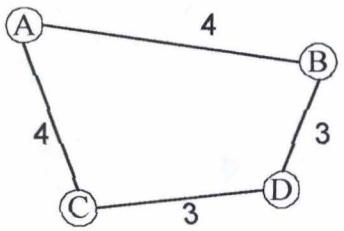

Figure 3.18 : Solution finale

L'itération suivante ne permet plus de trouver des transformations qui réduisent le coût du réseau de transit. L'algorithme se termine avec la solution d'optimum local de la *Figure 3.18* avec un coût de 14 unités. Dans le cas de cet exemple, cet optimum local est en même temps l'optimum global.

En utilisant cette méthode, le réseau résultant ne possède éventuellement qu'une solution d'optimum local. Pour ne pas rester bloqué sur cette solution locale, on va essayer de "surmonter" cet optimum. Ainsi, une fois que l'optimum local est trouvé, tout le processus est répété en recommençant par la procédure de départ et en déterminant une nouvelle solution initiale de manière aléatoire. Ensuite, la procédure d'optimisation est de nouveau mise en route.

La génération de solutions initiales aléatoires permet de dépasser les solutions d'optimum local (*Figure 3.19*) et de trouver plusieurs solutions. Cette recherche aléatoire permet d'espérer tomber sur des solutions proches de l'optimum global.

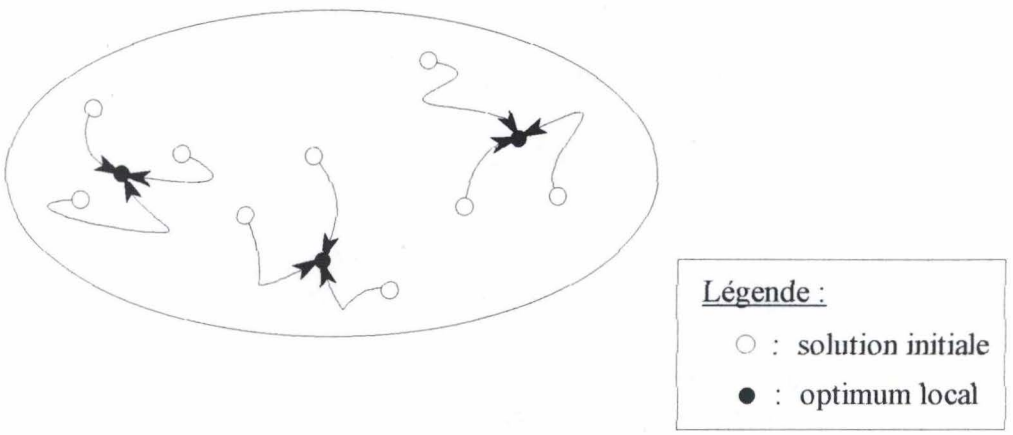

Figure 3.19 : Dépassement de minima locaux

Un inconvénient de cet algorithme est qu'on n'est pas sûr de trouver l'optimum global, puisqu'on part d'une solution initiale aléatoire et que les transformations ne permettent pas de tomber sur toutes les configurations possibles.

Algorithme dual de l'arbre minimal

L'algorithme de l'arbre minimal (cf. [PRI57]) est un algorithme qui vise à construire un arbre de coût minimal. Autrement dit, il relie des points par des lignes de coût minimal en ajoutant à chaque fois la ligne dont le coût est le plus petit. Dès que tous les points sont reliés entre eux, l'algorithme s'arrête.

Malheureusement, cette méthode appliquée à un réseau de transit ne permet pas de vérifier le critère de fiabilité, c'est-à-dire que la solution de cet algorithme n'est pas un réseau de connectivité d'ordre deux.

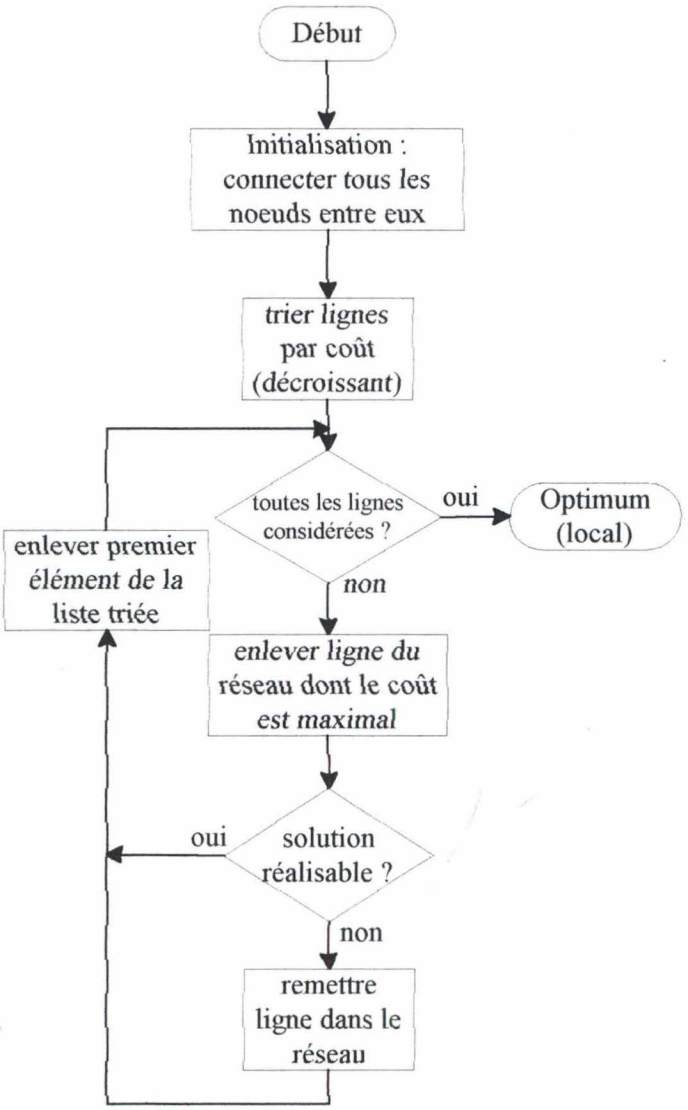

Figure 3.20 : Algorithme dual de l'arbre minimal

Sur base de cette approche, on peut utiliser une méthode alternative pour trouver un réseau réalisable : le dual de l'algorithme de l'arbre minimal. Il poursuit le même objectif (celui de trouver un arbre minimal), mais attaque le problème d'une autre manière : au lieu d'ajouter les lignes de coût minimal, on les retire (*Figure 3.20*).

L'algorithme procède de la manière suivante : on relie d'abord chaque noeud à tout autre noeud par une ligne et on établit la matrice des coûts de toutes ces lignes. Comme il y a une ligne entre tous les noeuds, la matrice des coûts se calcule en fonction du trafic moyen direct entre les noeuds. En plus, dans ce cas de base, le réseau est réalisable (si le nombre de noeuds est supérieur à deux).

Reprenons le même exemple que celui de la section précédente. La *Figure 3.21* montre un réseau à quatre noeuds dont la matrice des coûts est donnée dans le *Tableau 3.4*.

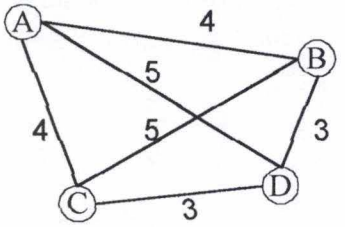

Figure 3.21 : Solution initiale

Lorsqu'on relie tous les noeuds, on obtient le réseau de la *Figure 3.21* ayant le coût suivant :

$$C_{total} = C_{AB} + C_{BD} + C_{CD} + C_{AC} + C_{AD} + C_{BC}$$
$$= 4 + 4 + 3 + 3 + 5 + 5 = 24$$

Ensuite, on enlève la ligne ayant le coût le plus élevé. Il faut alors vérifier si le réseau est toujours réalisable. Si c'est le cas, on accepte la solution et on continue sur cette solution. Dans le cas contraire, on ignore tout simplement l'enlèvement de la ligne et on repart sur l'ancienne solution.

Enlevons donc la ligne dont le coût est le plus élevé. Comme on a le choix entre AD et CB, pour lesquelles le coût vaut 5 unités, éliminons par exemple AD. On obtient alors un réseau dont le coût est inférieur au coût précédent :

$$C_{total} = C_{AB} + C_{BD} + C_{CD} + C_{AC} + C_{BC}$$
$$= 4 + 4 + 3 + 3 + 5 = 19$$

Comme le réseau est toujours réalisable, l'algorithme va accepter cette nouvelle solution.

A l'itération suivante, on enlève de nouveau la ligne de coût maximal parmi toutes les lignes encore non-examinées. On s'arrête lorsqu'on a examiné toutes les lignes du réseau. La solution sera un réseau de transit de coût localement minimal.

A la deuxième itération de notre exemple, on peut enlever la ligne CB qui a également un coût de 5 unités. De nouveau, le réseau reste réalisable ce qui nous permet d'accepter cette solution qui est représentée à la *Figure 3.18*. Le coût tombe à 14 unités. Ensuite, l'algorithme élimine la ligne AB (ayant un coût de 4 unités), dans quel cas le réseau n'est plus réalisable. On va donc ignorer cette opération et essayer d'enlever consécutivement les lignes AC, CD et BD. Ces suppressions font que le réseau devienne non-réalisable; on doit donc également les ignorer. Ainsi, la solution finale est donnée par le réseau de la *Figure 3.18*; elle correspond à celle trouvée à l'aide de l'algorithme de Steiglitz.

L'algorithme dual de l'arbre minimal s'adapte donc très bien à la recherche d'un réseau de transit "optimal" en vérifiant en même temps le critère de fiabilité. Même s'il n'y a pas de preuve que les solutions trouvées par cet algorithme sont des optima globaux, on peut prendre cette approche comme heuristique. De plus, on peut espérer que l'heuristique du dual trouve la solution optimale ou une solution assez proche de celui-ci.

Cut Saturation Algorithm

Le "Cut Saturation Algorithm" (cf. [GER74]) est un algorithme d'optimisation d'un réseau existant. Il vise à améliorer les performances et à optimiser les coûts du réseau existant. Pour augmenter les performances, il vise à optimiser le "throughput". Le "throughput" (exprimé en bps) représente la moyenne de tous les trafics moyens par ligne.

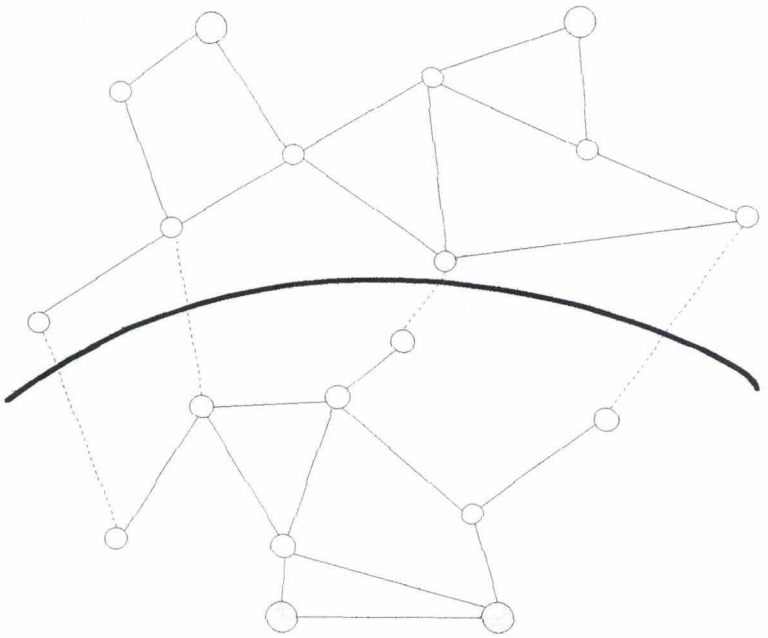

Figure 3.22 : Méthode de découpe en deux sous-réseaux

L'optimisation de la topologie se fait par une méthode de découpe en deux sous-réseaux qui permet de réduire le trafic sur les lignes les plus chargées du réseau (*Figure*

3.11). Cet algorithme trouve un réseau de coût minimal pour un taux d'occupation (intensité de trafic) donné et vérifiant les contraintes de temps de réponse et de connectivité. La méthode est mise en oeuvre par les cinq étapes suivantes :

Routage

L'algorithme de routage détermine le flux de communication d'un réseau afin de trouver le meilleur chemin entre deux noeuds en minimisant le temps de réponse moyen.

Détermination du Saturated Cutset (découpage en deux sous-réseaux)

Après application de l'algorithme de routage, les lignes sont ordonnées en fonction de leur taux d'utilisation. Plusieurs lignes sont coupées, en partant de la ligne dont l'intensité de trafic est la plus élevée et en continuant par ordre décroissant d'intensité de trafic. Cette coupure de lignes s'arrête dès que le réseau est découpé en deux sous-réseaux. L'ensemble de ces lignes, c'est-à-dire le plus petit ensemble de lignes qui déconnecte le réseau en sous-réseaux, est appelé "Saturated Cutset".

Etape d'insertion de lignes

Cette procédure réintroduit des lignes entre les deux sous-réseaux en réduisant le trafic sur les lignes précédemment coupées. Dans ce but, l'algorithme prévoit d'insérer la ligne la moins coûteuse qui relie les deux sous-réseaux. On considère qu'une ligne est coûteuse lorsque le rapport entre son coût et son taux d'occupation effectif est élevé. Cette ligne peut seulement relier les noeuds de chaque côté du "Cutset" qui sont assez loin les uns des autres pour connecter le réseau de manière efficace, c'est-à-dire pour que les nouvelles lignes ne soient pas tout de suite surchargées. Dans ce but, les noeuds qui délimitent la nouvelle ligne doivent se situer à au moins deux noeuds du "Cutset". Dans le cas de la *Figure 3.11*, seuls les noeuds grisés pourront être reliés.

Etape d'élimination de lignes

Cette étape de l'algorithme retire les lignes qui donnent lieu à une topologie dense. Autrement dit, la ligne dont le coût est le plus élevé et dont le taux d'occupation est le plus faible est retirée du réseau.

Etape de perturbation

Cette étape permet de réarranger les lignes du réseau afin de réduire les coûts. Si le "throughput" tombe en dessous de la borne inférieure (fixée au début), l'étape d'addition a pour effet d'augmenter le "throughput". Si le throughput devient plus grand que la borne supérieure, l'élimination des lignes a pour effet de diminuer le throughput. Lorsque le taux d'utilisation reste dans ces bornes, on applique successivement l'insertion et l'élimination de lignes.

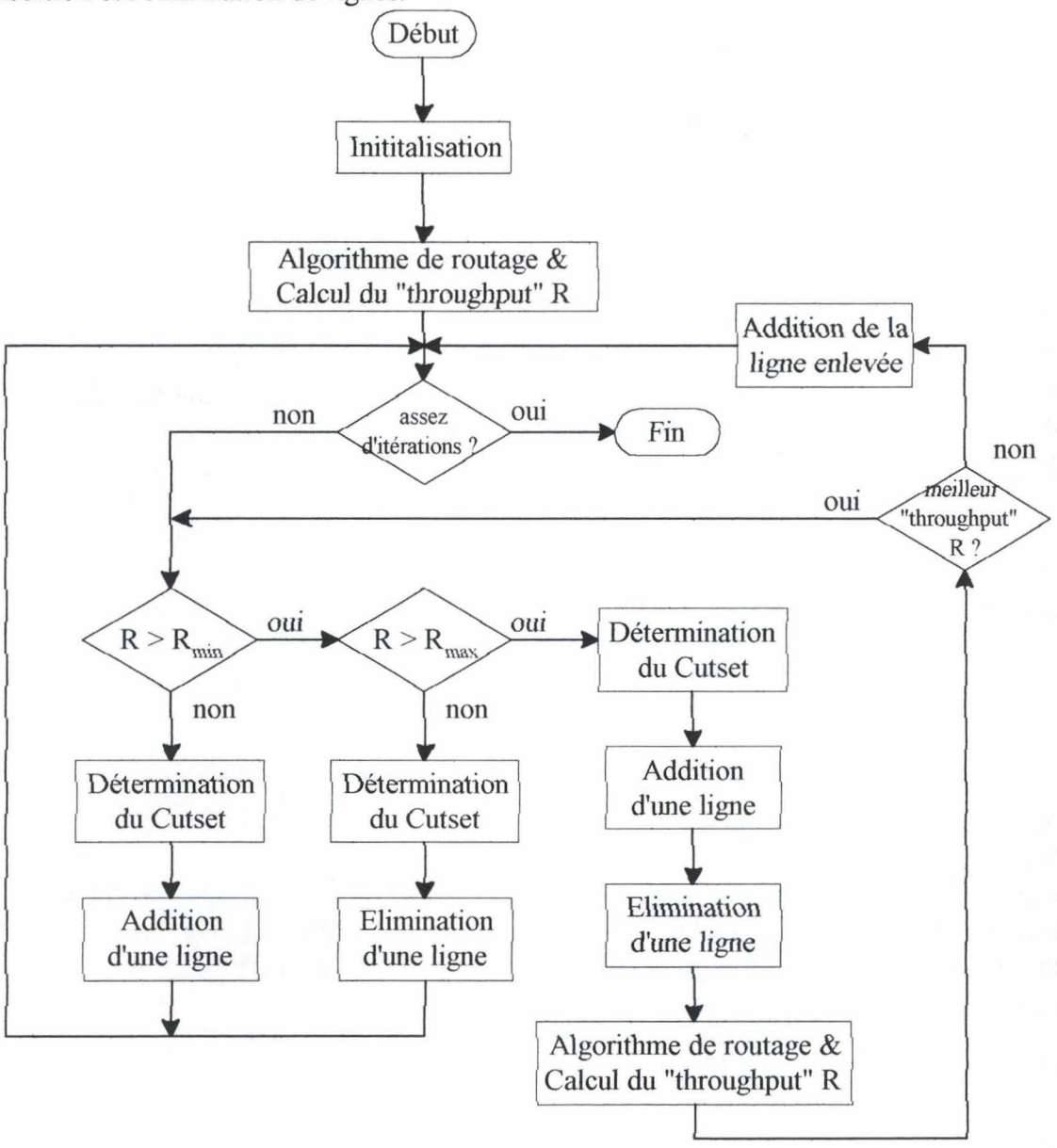

Figure 3.23 : Organigramme du Cut Saturation Algorithm

La *Figure 3.23* présente l'organigramme de l'algorithme de la découpe en sous-réseaux. Elle montre l'étape principale, la perturbation, qui utilise les quatre autres étapes. Comme ce type d'algorithme s'applique à des réseaux de grande taille - dû à la distance de deux noeuds du Cutset lors de l'étape d'insertion. C'est pour cela qu'une

illustration avec un exemple prendrait trop de temps et sortirait du cadre de ce mémoire. Le lecteur intéressé peut néanmoins trouver un exemple dans [GER74].

3.5. Réseau distribué

Comme déjà dit, le problème global est souvent divisé en sous-problèmes. Il faut alors une méthode ou un algorithme qui combine ces sous-problèmes afin de résoudre le problème de la conception du réseau complet. Un tel algorithme doit englober les quatre premières phases de la méthode décrite dans la section 3.1 (page 39). Cet algorithme constitue en quelque sorte le squelette d'un algorithme de recherche d'un réseau distribué de coût optimal puisqu'il fait appel à différents sous-algorithmes.

Le "Simulated Annealing" (recuit simulé - cf. [KIR83]), est un algorithme d'optimisation stochastique (non-déterministe). C'est un algorithme qui a été conçu pour simuler le processus de cristallisation en physique thermodynamique. Plus tard, on a appliqué cet algorithme à des problèmes de nature différente et compliqués à modéliser. Etant donné la complexité du problème de conception de réseaux, cet algorithme est un des rares qui puisse être appliqué au problème présent. Nous allons esquisser les points principaux de cette approche. Une description détaillée suit dans le chapitre suivant. On expliquera d'abord le principe physique, puis la formulation de l'algorithme du recuit simulé basé sur ce principe et finalement l'application aux réseaux.

Principe de base

Un corps solide est soumis à des températures très élevées, jusqu'à ce qu'il se trouve dans l'état liquide et ensuite on le laisse refroidir lentement. Ce refroidissement implique que des structures cristallines régulières se forment à partir de l'état liquide du corps solide. Lors de cette procédure, le solide parcourt différents états pour finalement aboutir dans un état cristallin. L'énergie complète du solide, laquelle est en physique classique une fonction de la position et des impulsions des atomes, décroît régulièrement lorsque la température diminue. A la fin, le corps solide se trouve dans un état d'énergie minimale.

On a pu modéliser ce processus et des applications dans d'autres domaines se sont présentées. On a également trouvé une relation entre la fonction d'énergie d'un solide et la fonction de coût d'un problème d'optimisation. Cette analogie existe si les conditions suivantes sont vérifiées :

- Les solutions du problème d'optimisation doivent être représentées par des états dans un espace d'états;
- Ces états doivent être évalués par une fonction de coût global;

- Il faut définir des transitions entre états.

Les avantages du recuit sont surtout basés dans le fait que le processus est réalisable de manière indépendante du problème. Face à cet avantage, il y a un inconvénient : le recuit est un processus stochastique non-déterministe et il n'y a donc pas de garantie de trouver l'optimum global.

Formulation de l'algorithme

L'algorithme du Simulated Annealing peut être décrit de la manière suivante : La première étape initialise la température à une valeur assez élevée et choisit aléatoirement un état dans l'espace des états comme solution initiale au problème d'optimisation. Cette température doit généralement être trouvée par des expériences.

Partant d'une solution, le mécanisme génère une solution voisine (un état voisin) dans l'espace d'états. La différence d'énergie détermine l'acceptation du nouvel état. Autrement dit, si l'énergie a diminué on accepte toujours l'état. Si par contre l'énergie n'a augmenté qu'un tout petit peu, on accepte quand même l'état lorsqu'on peut espérer trouver un état voisin qui permettra de diminuer l'énergie par après.

Ce processus itère à température constante jusqu'à ce que la différence d'énergie entre les états ne diffère plus que légèrement. Ceci signale l'équilibre thermodynamique, c'est-à-dire un état d'énergie presque constant pour une température donnée. Après acceptation du nouvel état, l'algorithme fait décroître la température pas à pas en générant à chaque fois un nouvel état à partir du précédent. Ainsi, il est raisonnable de parcourir l'espace d'états au début d'une simulation à grands pas taille et de diminuer le rythme vers la fin du processus de refroidissement. Après dépassement d'une température minimale l'algorithme se termine.

Deux fonctions sont essentielles pour l'algorithme du recuit simulé : la fonction qui calcule la nouvelle température après la stabilisation et la fonction qui génère les nouvelles solutions à une température fixe.

La complexité de l'algorithme dépend d'une part du nombre de pas qui sont nécessaires pour parcourir la plage de températures et d'autre part du nombre de pas nécessaires à l'obtention de l'équilibre thermodynamique pour une température fixée.

Application aux réseaux

Le principe de l'algorithme reste le même lorsqu'on l'applique au problème de la conception de réseaux. La température détermine l'acceptation d'une solution. Remarquons que la température du phénomène physique n'a pas de signification dans la conception des réseaux, mais nous allons expliquer le rôle de ce paramètre dans le chapitre suivant. Comme on l'a vu ci-dessus, l'algorithme effectue des transformations

pour une température donnée jusqu'à ce qu'il obtienne l'équilibre thermodynamique. Après, on diminue la température - d'abord de grandes diminutions, puis des petites. Ce paramètre doit donc être choisi de manière telle que l'algorithme accepte au début beaucoup de transformations et de moins en moins lorsque le paramètre décroît. Il est clair que ce paramètre dépend du problème posé et qu'il doit être trouvé par des tests. Une configuration (topologie) quelconque représente un état dans le problème physique. Finalement, la fonction de coût du réseau complet correspond à l'énergie.

Ainsi, l'algorithme s'écrit de la façon suivante. D'abord, on met le paramètre (température) à une valeur initiale (trouvée par des tests) et on choisit une topologie (état) initiale aléatoire. On génère alors une solution voisine par une légère modification. Cette modification pourrait par exemple être le changement de la capacité d'une ligne, l'ajout ou le retrait d'une ligne ou le déplacement d'un noeud, etc ... Malheureusement, on a constaté lors de l'implémentation de ces modifications que l'utilisation de toutes ces transformations complique la conception d'un réseau. Cela fait que l'algorithme trouve rarement une solution acceptable. On a alors choisi d'accepter seulement des transformations sur les noeuds et de chercher les interconnexions de ceux-ci (les lignes) par un algorithme des réseaux de transit.

La nouvelle topologie (nouvel état) est acceptée lorsque le coût du réseau (énergie) a diminué ou lorsqu'on peut espérer trouver une topologie voisine permettant de diminuer le coût. Ceci est effectué à paramètre constant tant qu'on n'obtient pas l'équilibre, c'est-à-dire tant qu'une diminution du coût reste possible.

Après avoir accepté la nouvelle topologie, on fait diminuer le paramètre et on génère une nouvelle topologie. Le procédé décrit ci-dessus est alors appliqué jusqu'à ce qu'on dépasse le seuil minimum du paramètre (température minimale).

3.6. Comparaison des algorithmes

Dans les sections précédentes, on a présenté plusieurs algorithmes qui contribuent à la conception de réseaux. Chacun de ces algorithmes a ses avantages et ses inconvénients. Comme ce mémoire est accompagné d'un programme qui effectue cette conception, il faut choisir le ou les algorithmes à implémenter parmi ceux expliqués ci-dessus. Essayons donc de voir lequel correspond le mieux à l'objectif de ce mémoire.

Les algorithmes d'insertion et d'élimination de Bahl et Tang sont des algorithmes de conception de réseaux centralisés. Ce sont des algorithmes qui ont prouvé leur efficacité dans de nombreuses applications. Or, comme ces algorithmes vont concevoir un réseau centralisé, ce type de réseau ne nous intéresse pas dans le cadre de l'implémentation d'un algorithme. Mais comme ce sont deux algorithmes classiques, il faut en parler.

Le deuxième type de réseau qu'on a proposé était celui des réseaux distribués. Ce sont précisément ces réseaux qui suivent l'objectif de ce mémoire. On a vu qu'il y a deux manières de concevoir des réseaux distribués.

Une possibilité consiste à prendre une seule heuristique pour la conception de réseaux puisqu'on a très vite vu qu'il était impossible de résoudre le problème par des méthodes analytiques. Malheureusement, une heuristique couvrant tous les aspects de la conception (des réseaux d'accès et des réseaux de transit) serait toujours très complexe.

L'autre possibilité consiste à combiner les algorithmes des réseaux d'accès local (algorithmes de la section 3.3) avec ceux du réseau de transit (algorithmes de la section 3.4). L'avantage de cette méthode est de pouvoir décomposer le grand problème de la conception de réseaux en plusieurs petits problèmes. Nous avons vu une méthode qui fait cette décomposition. Pour chaque point de cette démarche, nous avons proposé certains algorithmes qui couvrent le sujet.

Un algorithme très intéressant permettant de trouver les positions des noeuds est l'algorithme de la recherche des centres de gravités. Malheureusement, il peut parfois privilégier des positions de noeuds qui ne reflètent pas vraiment le trafic du réseau.

L'algorithme des réseaux d'accès local est l'algorithme qui sert à placer des lignes entre hôtes et noeuds. Il est très simple et trouve toujours les lignes de distance minimale.

L'algorithme de Steiglitz est un algorithme qui cherche les lignes entre noeuds, mais il a un inconvénient : c'est un algorithme de recherche locale. Il cherche donc des solutions par hasard qui subissent des transformations. On accepte seulement une nouvelle solution si la transformation fait diminuer le coût. Ainsi, l'algorithme exploite des minima locaux. On espère alors que le minimum global se trouve parmi ces minima.

L'algorithme dual de l'arbre minimal connecte également les noeuds entre eux par des lignes. Ce n'est pas - comme les autres algorithmes d'ailleurs non plus - un algorithme analytique, mais une heuristique. Le grand avantage de cet algorithme est qu'il trouve très rapidement une solution.

Quant au Cut Saturation Algorithm, il vise à optimiser un réseau existant. De plus, il essaie plutôt de réduire la charge des lignes et en même temps de diminuer les coûts. Dans le but de diminuer cette charge, on peut ajouter des lignes entre noeuds qui se trouvent au moins à deux noeuds du Cutset (cf. section 3.4).

L'algorithme du recuit simulé est un algorithme qui a déjà fait preuve de bons résultats pour des problèmes complexes. Comme notre problème de conception de réseaux est très complexe, c'est un des rares algorithmes qui peut espérer aboutir à de bons résultats. Son plus grand avantage est qu'il peut dépasser les minima locaux. C'est la raison pour laquelle nous allons utiliser le recuit simulé comme algorithme englobant les différentes étapes de la méthode décrite dans ce chapitre. Le recuit simulé va diriger la recherche vers la solution optimale en faisant appel aux autres

algorithmes. Dans le chapitre suivant, nous allons expliquer quels algorithmes nous allons utiliser dans les sous-problèmes et aussi pourquoi nous choisissons ceux-là.

En conclusion, on peut dire que le problème de la conception de réseaux comme ceux présentés dans ce mémoire est un problème très complexe. On trouve certains algorithmes dans la littérature, mais la plupart a été développée pour des problèmes spécifiques. Il faut donc essayer de combiner les algorithmes les plus simples afin de résoudre le problème de la conception. Pour les raisons déjà mentionnées ci-dessus, l'algorithme du recuit simulé nous semble être l'algorithme le plus adéquat pour atteindre notre objectif : la conception de réseaux distribués.

Chapitre 4

Description détaillée du recuit simulé

L'idée de l'algorithme du recuit simulé est basée sur un phénomène physique : le recuit. Le recuit (en anglais : annealing) est en fait un processus de refroidissement d'un corps solide en état liquide à température élevée afin d'obtenir une forme cristalline du solide. Le recuit a déjà été analysé et modélisé aux années 50 par Metropolis (cf. [MET53]). Au début des années 80, Kirkpatrick (cf. [KIR83]) a trouvé une analogie entre ce concept du recuit et des problèmes d'optimisation.

4.1. Le recuit (Annealing)

Principe physique

En métallurgie, le terme de recuit est connu comme un processus thermodynamique qui consiste à chauffer un corps solide (d'habitude un produit métallurgique) à une température suffisamment élevée et puis à opérer un refroidissement lent. Si la température est assez élevée, le corps solide se trouve à l'état liquide. En diminuant la température lentement, le corps devient de plus en plus solide. Ce processus permet une restructuration des particules à l'intérieur du corps solide de manière telle que le résultat final du processus donne un état stable du corps. Dans ce cas, le corps solide prend une structure cristalline régulière. Cet état est appelé état de base du corps solide.

D'une part, lorsque le corps se trouve à l'état liquide, toutes les particules du corps solide sont arrangées de manière aléatoire : il n'y a pas de structure à l'intérieur du corps. D'autre part, en état de base, les particules s'arrangent dans une structure extrêmement rigide et le corps prend une structure cristalline régulière. Un système atteint cette rigidité lorsque son énergie est minimale.

Lors du processus du recuit, un tel état de base du solide peut seulement être obtenu si la température de départ est assez élevée et si le refroidissement est fait de manière lente et régulière. La valeur initiale de la température et la lenteur du refroidissement dépendent bien sûr de la nature du corps solide. Lors du refroidissement, l'énergie totale du corps solide décroît de manière régulière. En fait, on refroidit lentement la substance pour assurer l'équilibre thermodynamique. Pour une température donnée, le système se trouve en équilibre lorsque la différence d'énergie entre deux états voisins est très petite. Un état est la position des particules dans le corps solide. Un état voisin est une nouvelle position des particules après leur mouvement.

La technique opposée au recuit simulé est celle de la trempe, qui consiste à abaisser très rapidement la température du corps solide. Dans ce cas, on obtient une structure amorphe : le corps se trouve dans un état peu stable, appelé état métastable. Cet état correspond à un minimum local de l'énergie.

Avec la technique du recuit, le refroidissement du corps solide provoque une transformation d'un état en désordre vers un état ordonné. Par contre, la technique de la trempe finit par figer un état désordonné. La *Figure 4.1* illustre les résultats des deux méthodes.

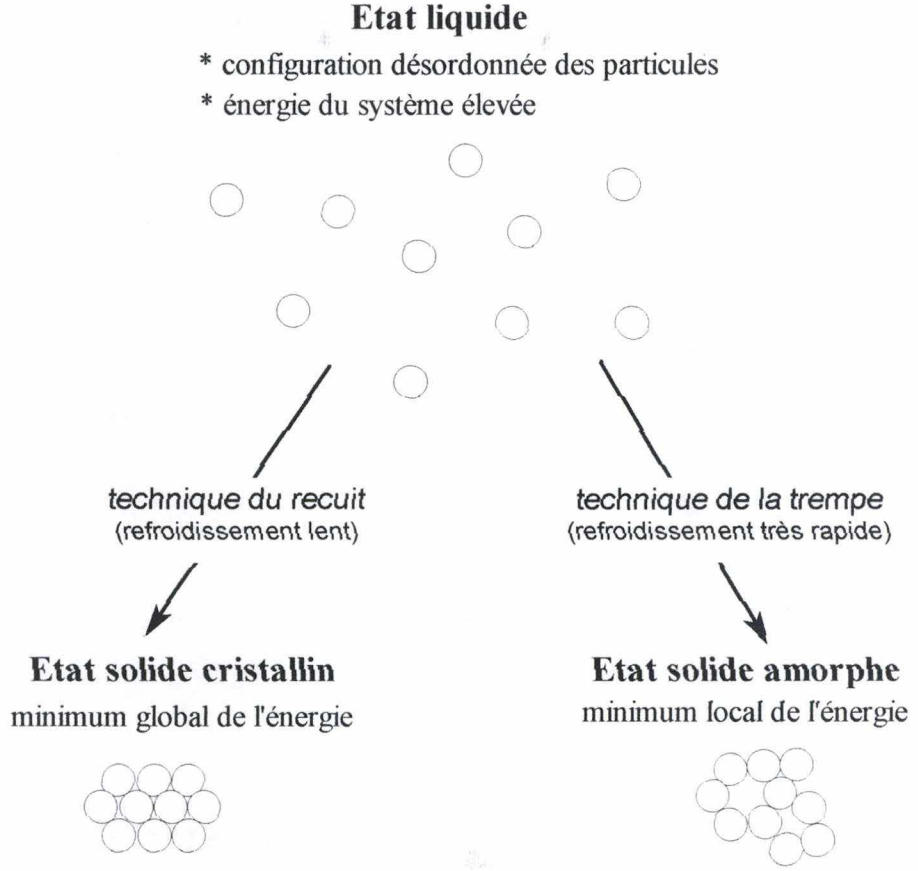

Figure 4.1 : Comparaison des techniques du recuit et de la trempe

Modélisation

Déjà en 1953, Metropolis (cf. [MET53]) a modélisé avec succès le processus physique du recuit et a implémenté des méthodes de simulation par ordinateur. Il a introduit un algorithme simple pour simuler le phénomène du recuit décrit ci-dessus. L'algorithme introduit par les auteurs tombe dans la catégorie des méthodes de Monte-Carlo. Il génère une suite d'états représentant l'état du solide à une certaine température. Chaque état possède une certaine énergie. La suite générée mène finalement à l'état terminal (état de base) qui est en fait l'état d'énergie minimale. Les états intermédiaires sont générés de la façon suivante :

Considérons un état courant i du corps solide avec une énergie E_i. Pour trouver l'état suivant j ($= i+1$), on met en oeuvre la procédure suivante :

1. On perturbe légèrement la configuration pour transformer l'état i courant en un état voisin j par une petite déformation; généralement en déplaçant une particule. Puis, on calcule l'énergie E_j de ce nouvel état voisin j.

2. Si la différence d'énergie $\Delta E = E_j - E_i$ est plus petite ou égale à 0 (c'est-à-dire s'il y a une diminution de l'énergie), l'état j est accepté comme état courant.
3. Si la différence d'énergie ΔE est plus grande que 0, l'état j est accepté avec une probabilité donnée par

$$\exp\left(\frac{E_i - E_j}{k_B T}\right) \qquad (4.1.)$$

où T désigne la température (en degrés Kelvin) à laquelle le processus se trouve et où k_B est une constante physique connue sous le nom de constante de Boltzmann.

Lorsqu'on atteint l'équilibre thermodynamique pour un niveau de température, c'est-à-dire si la variation de l'énergie est très petite, on fait diminuer cette température lentement. Remarquons qu'en diminuant la température, la probabilité d'accepter un nouvel état donnée par l'expression (4.1.) est plus faible pour une même différence d'énergie. Pour garantir cet équilibre dans l'algorithme, il faut générer un grand nombre de transitions pour chaque palier de température.

La règle (4.1.) est connue comme règle d'acceptation de Metropolis (ou critère de Metropolis) et les algorithmes qui se basent sur ce critère sont appelés algorithmes de Metropolis.

4.2. L'algorithme du recuit simulé (Simulated Annealing)

On expliquera ici l'analogie au phénomène physique du recuit avec des problèmes d'optimisation en général afin de mieux comprendre le fonctionnement du recuit simulé. Une analyse détaillée de l'algorithme du recuit simulé appliqué au cas de la conception de réseaux suivra à la section 4.3.

Analogie avec le recuit

En 1982, Kirkpatrick (cf. [KIR83]) a montré que l'algorithme de Metropolis permettait également de générer une séquence de solutions d'un problème d'optimisation combinatoire. Cette idée d'utiliser la technique du recuit en vue de traiter un problème d'optimisation a donné naissance à la méthode du recuit simulé. Elle consiste à introduire, en optimisation, un paramètre de contrôle, qui joue le rôle de la température. La "température" du système à optimiser doit avoir le même effet que la température du système physique : elle doit conditionner le nombre d'états accessibles et conduire à l'état optimal si elle est abaissée de façon lente et bien contrôlée (technique du recuit) et vers un minimum local si elle est abaissée brutalement (technique de la trempe).

L'analogie qui vient d'être mise en évidence par ces auteurs entre un problème d'optimisation combinatoire et un système physique constitué d'un grand nombre de particules en interaction peut être résumée par le tableau suivant :

Problème d'optimisation	**Système physique**
solution du problème	état du système
fonction objectif f	*énergie E*
paramètres du problème	"coordonnées" des particules
trouver la configuration optimale	trouver l'état d'énergie minimale

Tableau 4.1 : Analogie entre un problème d'optimisation et un système physique

En résumé, la nature sait parfaitement résoudre des problèmes d'optimisation complexe très semblables à ceux que rencontre par exemple un ingénieur qui cherche à modéliser et optimiser un système complexe.

En tenant compte de l'équivalence entre les différentes variables de ces deux problèmes, on peut déduire l'algorithme du recuit simulé :

> Soient *i* et *j* deux solutions d'un problème d'optimisation combinatoire ayant respectivement une fonction objectif - ou coût - *f(i)* et *f(j)*. En vertu de ce qui a été dit pour l'algorithme de Metropolis, le critère d'acceptation détermine si la solution *j* est acceptée à partir de la solution *i* en appliquant la probabilité d'acceptation suivante :

$$P_c\{\text{accepter } j\} = \begin{cases} 1 & \text{si } f(j) \le f(i) \\ \exp\left(\dfrac{f(i)-f(j)}{c}\right) & \text{si } f(j) > f(i) \end{cases} \qquad (4.2.)$$

> où *c* désigne le paramètre de contrôle.

La transformation d'une solution courante en une solution voisine est appelée transition. Le critère d'acceptation détermine si la solution est acceptée ou non.

L'algorithme du recuit simulé est représenté par l'organigramme suivant (*Figure 4.2*):

Comme la figure le montre, l'algorithme du recuit simulé possède une règle bien particulière : la règle d'acceptation de Metropolis. Cette règle autorise d'une part les solutions qui diminuent le coût du problème d'optimisation et d'autre part, elle accepte des perturbations (de petites augmentations du coût) jusqu'à un certain degré, limité par le paramètre de contrôle c. Au début de l'algorithme (si les valeurs du paramètre de contrôle c sont encore importantes) presque toutes les transitions sont acceptées, même si le coût augmente fortement. Au plus c décroît, au moins de changements (dégradations) sont tolérés. Si finalement c approche la valeur 0, on n'accepte plus que les transitions qui diminuent le coût du problème d'optimisation. Remarquons que si c = 0, l'algorithme du recuit simulé correspond à un algorithme de recherche locale.

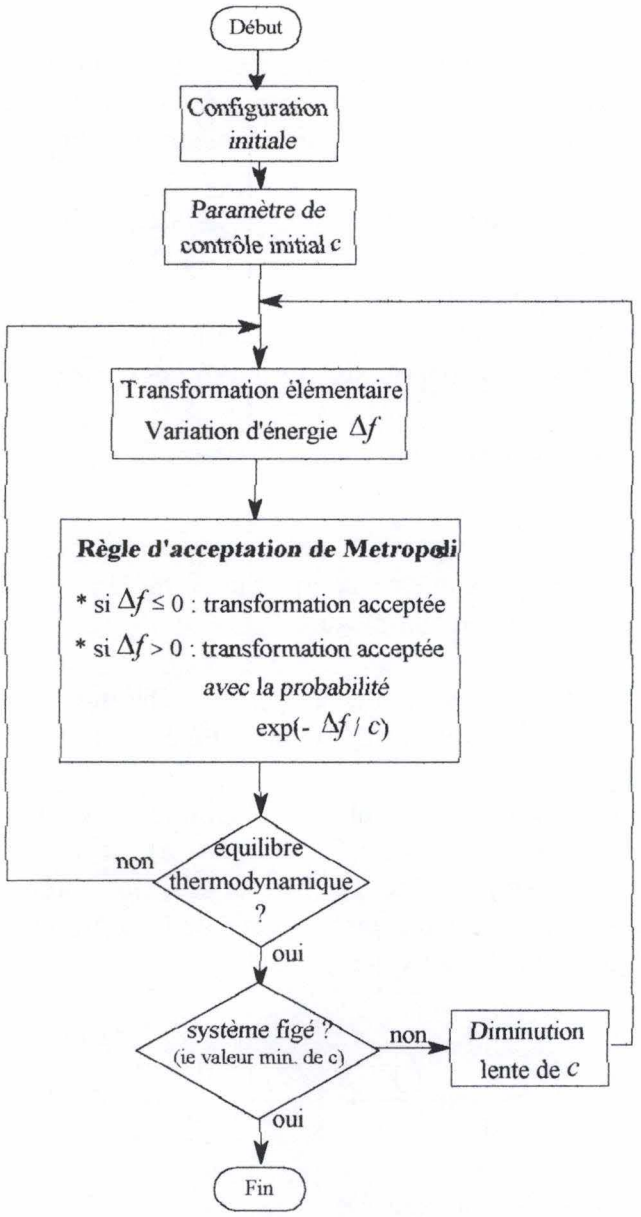

Figure 4.2 : Algorithme du recuit simulé

Contrairement aux algorithmes de recherche locale, ce comportement particulier du recuit simulé permet de sortir des minima locaux tout en restant simple (*Figure 4.3*).

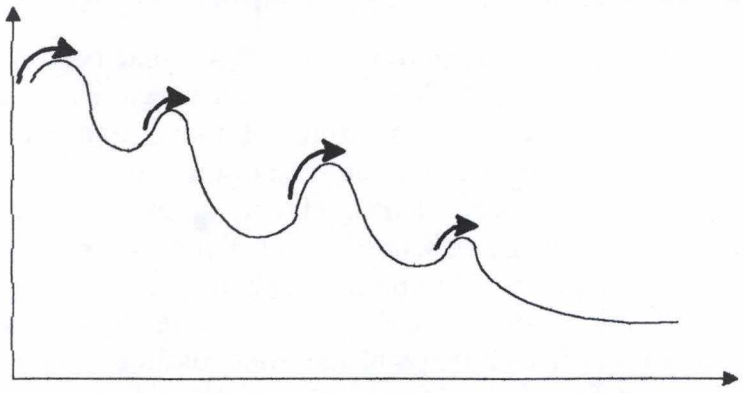

Figure 4.3 : Dépassement des minima locaux

Remarquons que la vitesse de convergence de l'algorithme est déterminée par le choix de la valeur initiale du paramètre de contrôle c, de sa loi de décroissance et par le nombre de transitions qu'on effectue à l'intérieur d'un palier du paramètre de contrôle.

Etude théorique

L'algorithme du recuit simulé a suscité de nombreux travaux théoriques pour les deux raisons suivantes : d'une part, il s'agit d'un algorithme nouveau dont il est nécessaire d'établir les conditions de convergence; d'autre part, la méthode comporte beaucoup de paramètres et de variantes dont il faut comprendre le mécanisme si l'on veut obtenir une efficacité maximale.

Les études théoriques sur le recuit simulé s'appuient généralement sur la description de l'algorithme comme une suite de chaînes de Markov. Dans cet algorithme, la succession des états (solutions) forme une chaîne de Markov, en ce sens que la probabilité de transition de la solution *i* à la solution *j* ne dépend que de ces deux solutions et pas des solutions antérieures à *i*. En d'autres termes, tout le passé du système est résumé par l'état courant (la solution courante).

Il a été montré par plusieurs auteurs (cf. [KIR83] et [SIA88]) que la convergence de l'algorithme du recuit simulé est assurée lorsque le paramètre de contrôle tend vers 0. Dans ce cas, l'algorithme génère une infinité d'états; la chaîne de Markov est donc infinie. Une telle chaîne aboutit à coup sûr au résultat optimal si elle est construite pour une valeur du paramètre de contrôle assez basse. Mais, ce résultat n'est d'aucune utilité pratique, car dans ce cas, l'approche de l'équilibre est très lente et peut même être exponentielle. Le formalisme des chaînes de Markov permet d'examiner sur le plan théorique la vitesse de convergence de l'algorithme. On peut montrer que cette vitesse est améliorée lorsque l'on part d'une valeur élevée du paramètre de contrôle et que l'on fait décroître celui-ci par paliers.

On aboutit généralement à une implémentation dans laquelle des séquences de chaînes de Markov sont finies. Ces chaînes de Markov sont générées en essayant continuellement de transformer une solution courante en une solution voisine en appliquant d'abord les transitions et puis le critère d'acceptation.

L'application correcte de l'algorithme du recuit simulé demande la spécification des points suivants :

1. Représentation précise du problème

Une description précise et correcte de la représentation du problème consiste à donner une "bonne" représentation de l'espace des solutions (l'espace des états) et une expression correcte de la fonction de coût. L'espace des solutions doit être tel que chaque solution est accessible par un nombre fini de transitions à partir d'un certain état courant. La fonction de coût doit être telle qu'elle reflète le mieux l'objectif de

l'optimisation : certaines transitions doivent être pénalisées par rapport à d'autres. Pour limiter le temps de calcul, il faut que la représentation du problème et la fonction de coût soient données par des expressions simples et faciles à manipuler.

2. Mécanisme de génération de transitions

Générer une solution voisine acceptable à partir d'une solution courante consiste à effectuer les trois étapes suivantes :

1. effectuer une transition sur la solution courante;
2. calculer la différence de coût entre la nouvelle solution et la solution courante;
3. accepter ou refuser cette solution en fonction de la règle d'acceptation de Metropolis.

La génération d'une transition est généralement conçue de manière telle que la nouvelle solution soit obtenue directement de la solution courante par simple réarrangement. Ces réarrangements simples doivent être calculés rapidement : ce sont par exemple des permutations, des échanges ou des inversions. Le calcul de la différence de coût entre la solution courante et la nouvelle solution est effectué, si possible, de manière incrémentale. Autrement dit, dans ce cas on ne tient compte que des différences résultant des réarrangements locaux. Pour la plupart des applications, c'est la façon la plus rapide de calculer la différence de coût.

L'acceptation de nouvelles solutions est basée sur le critère d'acceptation de Metropolis (cf. formule (4.2.)).

3. Programme de recuit (cooling schedule)

Pour qu'une méthode itérative soit acceptable, il faut que sa convergence soit assurée et pour que la méthode soit utilisable, sa vitesse de convergence doit être la plus grande possible. Dans le cas de la méthode du recuit simulé, la convergence dépend d'une part de l'espace des solutions et d'autre part du programme de recuit. L'espace des solutions a déjà été étudié ci-dessus. Le programme de recuit spécifie des paramètres qui contrôlent la convergence de l'algorithme.

Un programme de recuit (cooling schedule) spécifie :

- une séquence finie de valeurs du paramètre de contrôle

  * *une valeur initiale du paramètre de contrôle*
    Comme déjà mentionné, la valeur initiale du paramètre de contrôle doit être assez grande pour permettre à l'algorithme de se diriger dans toutes les directions et d'accepter toutes les transitions possibles. Il y a des procédures itératives (cf. [AAR89] et [SIA88]) qui permettent de trouver de "bonnes" valeurs initiales pour beaucoup de problèmes. Mais en général, il faut la trouver par des essais préliminaires. Au cours de tels essais, on calcule l'évolution du système pendant un temps limité et on acquiert de la sorte une connaissance de l'espace des

solutions, à partir de laquelle on peut déterminer la valeur initiale du paramètre de contrôle. Cette expérience préliminaire peut consister simplement à calculer la valeur moyenne de la variation d'énergie en maintenant le paramètre de contrôle à 0.

* *une loi décroissante du paramètre de contrôle*
  Généralement, on choisit de petits changements dans la valeur du paramètre de contrôle. Une fonction fréquemment utilisée pour diminuer cette valeur est donnée par

$$c_{k+1} = \alpha \cdot c_k \qquad (4.3.)$$

  où α est une constante < 1. Des valeurs typiques de α se situent entre 0,80 et 0,99.

* *une valeur finale du paramètre de contrôle*
  On arrête l'algorithme si le coût de la dernière solution de la chaîne de Markov n'a plus changé pour les dernières chaînes de Markov.

- un nombre fini de transitions à chaque valeur du paramètre de contrôle

  * *une longueur finie de chaque chaîne de Markov*
    Pour que la longueur des chaînes de Markov, qui détermine la durée des paliers de température (paliers de paramètre de contrôle), soit finie, on va se contenter d'atteindre le quasi-équilibre pour un palier du paramètre de contrôle. On a pu constater que, pour atteindre ce quasi-équilibre, il suffit d'un nombre fini de transitions. Or, comme les transitions sont acceptées avec une probabilité décroissante lorsque le paramètre de contrôle tend vers 0, la longueur des chaînes de Markov est limitée par une certaine constante. Cette constante doit de nouveau être trouvée par des tests préliminaires.

4.3. Application aux réseaux

Hypothèses

On a vu que le recuit simulé est un algorithme non-déterministe hérité d'un phénomène physique. Cet algorithme s'adapte bien à des problèmes d'optimisation et donc également au problème de conception envisagé dans ce mémoire. Avant de voir comment la conception d'un réseau peut être résolu par l'algorithme du recuit simulé, rappelons les données :

Le réseau qui fera l'objet de la conception est un réseau d'une entreprise privée et peut s'étendre à toute la Belgique. On considère que le réseau est un réseau à commutation par paquets et que le routage à travers le réseau est fixe.

De plus, on fait les hypothèses suivantes :

- la position des machines hôtes est fixée,
- le trafic moyen entre hôtes est donné par la matrice de communication,
- on propose un ensemble de positions dans lequel l'algorithme choisira celles où il va effectivement placer des noeuds,
- les hôtes sont directement liés aux noeuds; on n'accepte pas de lignes multipoints,
- les capacités des lignes sont des multiples de 64 Kbps et leur coût est déterminé par les tarifs du fournisseur.

L'algorithme de conception de réseaux vise à placer les noeuds et les lignes. Il en déduit la capacité et le coût des lignes dans le but de minimiser le coût global du réseau tout en respectant le critère de performance des 60 %.

On impose deux contraintes supplémentaires à l'algorithme du recuit simulé :

1. Pour éviter qu'un noeud soit isolé lorsqu'une ligne tombe en panne, on exige une connectivité d'ordre deux. Cette contrainte est vérifiée si chaque noeud possède au moins deux lignes qui mènent vers deux autres noeuds.
2. Etant donné que le coût n'est calculé que sur la location mensuelle des lignes et pas sur l'acquisition du matériel, il faut donner la possibilité de limiter le nombre de noeuds.

Nous allons maintenant essayer de voir comment l'algorithme du recuit simulé peut être mis en oeuvre. Appliquons les différents points de l'étude théorique de la section 4.2. au problème de conception d'un réseau.

Représentation précise du problème

Donnons d'abord la représentation de l'espace des solutions. Une solution est constituée des éléments suivants :

- un ensemble d'hôtes placés à des endroits bien précis et prédéfinis,
- un ensemble de noeuds qui se trouvent également à des endroits très précis. Ces endroits ont été choisis parmi un ensemble de positions connues,
- des lignes entre hôtes et noeuds : chaque hôte est relié à un seul noeud,
- des lignes entre noeuds : chaque noeud est relié à au moins deux autres noeuds.

Rappelons que les lignes sont caractérisées par les coordonnées de leurs extrémités (et donc par leur longueur selon la classification Belgacom en parties zonales et interzonales) et par leur capacité. Cette capacité se déduit indirectement de la matrice de communication (voir chapitre 2). Pour connaître la capacité des lignes entre noeuds, il faut en plus avoir une matrice de routage. La matrice de communication et la matrice de routage influencent donc indirectement l'espace des solutions.

L'expression de la fonction de coût du problème de conception est en fait le coût total du réseau. Le calcul de ce coût a été étudié en détail au chapitre deux.

Mécanisme de génération de transitions

La solution initiale est générée de manière aléatoire et doit être réalisable (c'est-à-dire de connectivité d'ordre deux). Le passage d'une solution à une solution voisine se fait par la génération de transitions. Dans notre cas, on peut envisager huit transitions différentes :

1. le déplacement d'une ligne qui relie un hôte et son noeud,
2. l'ajout d'une ligne entre deux noeuds entre lesquels il n'y a pas encore de liaison directe,
3. le retrait d'une ligne entre deux noeuds,
4. le déplacement d'une ligne entre noeuds,
5. l'échange de deux lignes ("X-change" utilisée dans l'algorithme de Steiglitz),
6. l'ajout d'un noeud,
7. le retrait d'un noeud,
8. le déplacement d'un noeud.

Le mécanisme de génération de transitions décrit ci-dessus était la première approche qu'on a envisagé d'implémenter dans le programme qui accompagne ce mémoire. Le but était de concentrer la recherche de la topologie optimale (réseau optimal) à l'intérieur de l'algorithme du recuit simulé. Tous les changements et toutes les modifications se faisaient par des transitions du recuit simulé.

Malheureusement, cette approche s'est avérée trop complexe. Nous allons tenter d'expliquer pourquoi :

L'approche

Conformément aux explications concernant les différentes approches dans la conception de réseaux, il faut dire que cette approche rentre dans la catégorie des algorithmes généraux. Comme déjà vu, ces algorithmes attaquent le problème tout entier sans diviser la tâche de la conception en deux : la conception des réseaux d'accès local et des réseaux de transit. En implémentant l'algorithme du recuit simulé de la façon décrite ci-dessus (concevoir tout le réseau par un seul algorithme), on amplifie la complexité de la conception de réseaux.

Une première amélioration à faire est d'éliminer la première transition dans la liste ci-dessus. Cette transition déplace la connexion d'un hôte à un noeud vers un autre noeud. Or, nous avons vu dans la section 3.3 du chapitre précédent, qu'il faut tout simplement connecter l'hôte au noeud le plus proche. Lorsqu'on divise le problème de conception en deux parties, on peut utiliser l'algorithme (simple) du réseau d'accès local (cf. chapitre précédent).

Le fait de chercher les lignes entre hôtes et noeuds par un algorithme distinct évite d'appliquer des transitions pour quelque chose d'aussi simple que la connexion d'un hôte au noeud le plus proche.

Transitions complexes

Les sept transitions restantes (2 à 8) sont assez élémentaires en soi, mais l'algorithme doit vérifier le critère de fiabilité, c'est-à-dire qu'à tout instant, la solution doit être de connectivité d'ordre deux. Cette contrainte doit être vérifiée après n'importe quelle modification du réseau. C'est surtout l'implémentation de cette vérification qui a posé des problèmes. Afin de mieux comprendre, nous allons expliquer les problèmes rencontrés ci-dessous :

- Transition 2 : l'ajout d'une ligne. Avant d'ajouter une ligne entre deux noeuds, il faut vérifier qu'elle n'existe pas encore. Comme le réseau vérifie le critère de fiabilité avant l'application de la transition, l'ajout d'une ligne ne pose pas de problème au niveau de la connectivité. Cette transition est finalement simple à implémenter.
- Transition 3 : le retrait d'une ligne. Cette opération risque de diminuer l'ordre de connectivité du réseau. Une première idée était d'enlever seulement une ligne lorsque les deux noeuds à leurs extrémités ont toujours au moins deux autres lignes reliées à d'autres noeuds. Mais cette restriction n'est pas assez forte. Après quelques itérations, on court le risque d'isoler une partie du réseau du restant. De plus, cette restriction ne permet pas de satisfaire au critère de fiabilité, puisque celui-ci exige deux chemins distincts entre n'importe quelle paire de noeuds. La vérification du critère de fiabilité exige un calcul considérable et nécessite un petit algorithme. Donc, à chaque fois qu'on applique cette transformation, il faudrait également appliquer cet algorithme; d'où un nombre important de calculs supplémentaires.
- Transition 4 : le déplacement d'une ligne. Cette transition donne en fait le même résultat que la combinaison des deux transitions précédentes : l'ajout et le retrait d'une ligne. On a donc rencontré les mêmes problèmes que ceux des transitions 2 et 3.
- Transition 5 : l'échange de deux lignes (X-change). Avant d'appliquer cette transition, il faut vérifier que le X-change ne génère pas deux lignes qui existent déjà. Si on crée deux nouvelles lignes, la connectivité d'ordre deux est toujours respectée.
- Transition 6 : l'ajout d'un noeud. Il ne suffit pas d'ajouter tout simplement le noeud au réseau; il faut encore le relier à ce dernier. Comme il faut une connectivité d'ordre deux, on doit ajouter - en plus du noeud - deux lignes à deux autres noeuds distincts.
- Transition 7 : retrait d'un noeud. Cette opération a le plus de conséquences sur la structure du réseau et nécessite le plus de calculs pour satisfaire au critère de fiabilité. Lorsqu'on enlève un noeud, on déconnecte également les hôtes reliés à celui-ci. Donc, il faut les relier de nouveau à d'autres noeuds. Et lorsqu'on retire un noeud, on doit également retirer les lignes attachées précédemment au noeud en question. En retirant ces lignes, les noeuds de l'autre bout risquent de perdre leur connectivité d'ordre deux.
  Une première idée était de relier les noeuds deux à deux. Mais cela n'est possible que si le noeud retiré possède un nombre pair de lignes et si les nouvelles lignes n'existaient pas auparavant. Et que faire dans le cas contraire? Il faut encore une fois vérifier la contrainte de fiabilité pour tous ces noeuds qui perdent la connexion au noeud retiré.

- Transition 8 : le déplacement d'un noeud. C'est la seule transition sans vérification ultérieure puisque seule la longueur des lignes change. La connectivité est donc toujours garantie.

Comme on vient de le voir, il faut dans au moins 3 des 7 transitions vérifier explicitement par un algorithme supplémentaire si la connectivité d'ordre deux du réseau complet est toujours vérifiée. Comme ces calculs doivent être effectués à chaque transition, les performances de l'algorithme vont considérablement baisser. De plus, lorsque le critère de fiabilité n'est pas vérifié, on doit ignorer la transition. L'implémentation de cette approche a montré qu'un grand nombre de transitions est ainsi généré et refusé après. Ceci mène à beaucoup de calculs inutiles.

A tous ces problèmes s'ajoute un problème majeur qui est propre au problème de la conception de réseaux distribués : à chaque fois qu'on ajoute, enlève ou déplace une ligne (excepté pour la transition 8), il faut connaître la capacité de la ligne. Comme ces transitions modifient la structure du réseau, une partie des paquets qui véhiculent dans ce réseau, prend des chemins différents. Ceci implique qu'il faut recalculer le trafic non seulement sur une nouvelle ligne, mais sur toutes les lignes du réseau. Une simple modification peut donc changer le comportement de tout le réseau.

Un élément important dans le (re-)calcul du trafic est - comme déjà vu au chapitre deux - la matrice de routage. Il y a beaucoup d'algorithmes de routage dans la littérature. Nous allons présenter l'algorithme de routage utilisé dans le programme à la fin de ce chapitre.

Les calculs supplémentaires qu'il faut effectuer en plus des transitions mêmes, influencent - et peuvent même dégrader - les performances de l'algorithme. Ces calculs rendent l'algorithme encore plus complexe qu'il ne l'est déjà. Il faut donc trouver un moyen de simplifier l'algorithme du recuit simulé présenté sous la forme décrite ci-dessus. La façon de résoudre ce problème est la suivante. On essaye :
- de limiter le nombre de transitions et de grouper plusieurs transitions,
- d'éviter des calculs inutiles.

Nous allons donc "réorganiser" un peu la structure de l'algorithme du recuit simulé. L'approche va s'orienter vers la méthode de Sharma expliquée à la section 3.2 au chapitre précédent.

Méthode de Sharma et recuit simulé

Comme déjà mentionné ci-dessus, nous allons remplacer toutes les transitions décrites plus haut par l'approche de Sharma (cf. [SHA90]). Rappelons les cinq étapes de cette méthode:

1. choisir le nombre de noeuds,
2. chercher l'emplacement des noeuds,
3. relier les lignes entre hôtes et noeuds,
4. connecter les noeuds entre eux,
5. répéter les étapes 1 à 4 jusqu'à ce qu'on trouve une solution satisfaisante.

Au lieu d'appliquer les huit transitions, nous allons nous limiter à trois transitions de base :

a.) insérer un noeud,
b.) supprimer un noeud,
c.) déplacer un noeud.

Ces trois transitions couvrent les deux premières étapes de la méthode de Sharma: déterminer le nombre et l'emplacement des noeuds. A l'aide des algorithmes supplémentaires (résolvant les points 3 et 4 de la démarche ci-dessus), on va trouver la position des lignes.

L'algorithme des réseaux d'accès local permet de trouver les lignes entre hôtes et noeuds (étape 3). Comme cet algorithme relie chaque hôte au noeud le plus proche, c'est en même temps la ligne de coût minimal.

Dans le chapitre précédent, nous avons vu deux algorithmes de conception de réseaux de transit qui cherchent les lignes entre noeuds afin de trouver le réseau optimal. Comme nous allons utiliser ce type d'algorithme à l'intérieur de l'algorithme du recuit simulé, nous avons choisi le plus simple et le plus rapide des deux : l'algorithme dual de l'arbre minimal. En implémentant cet algorithme dans le recuit simulé on optimise déjà à l'intérieur de l'algorithme général. Ceci permet d'accélérer la recherche du réseau optimal - même s'il faut faire des calculs supplémentaires. L'organigramme de l'algorithme du recuit modifié par les points décrits ci-dessus est représenté par la *Figure 4.4*.

Un inconvénient reste tout de même. Après chaque transition, il faut calculer la différence de coût. Or, dans l'étude théorique de l'algorithme du recuit simulé, on propose de calculer le coût d'une manière itérative, c'est-à-dire de ne calculer que la petite différence que la transition engendre. Mais malheureusement, chaque génération d'une transition a ici pour effet de changer la répartition du trafic à travers tout le réseau. Cette nouvelle répartition modifie la capacité de chaque ligne et par conséquent le coût de la ligne, et finalement le coût du réseau entier. Ainsi, chaque fois qu'on effectue une transition, il faut calculer le coût du réseau complet et le comparer au coût de la solution précédente.

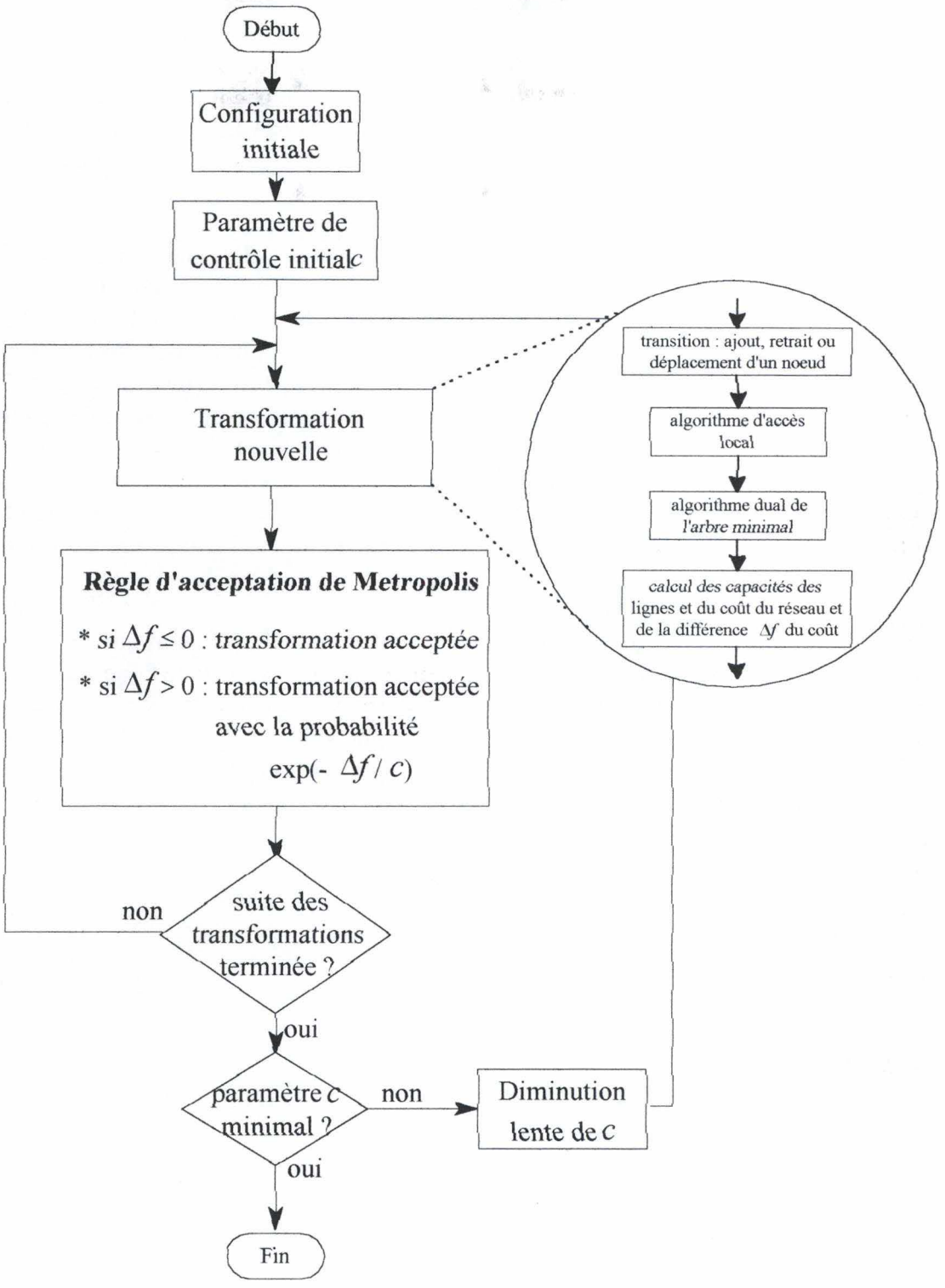

Figure 4.4 : Algorithme du recuit simulé appliqué aux réseaux

4.4. Algorithme de routage

Rappelons que dans notre cas le routage des paquets à travers le réseau est fixe. Ceci veut dire que les paquets envoyés d'un hôte source vers un hôte de destination prennent toujours le même chemin. En général, les paquets suivent le chemin le plus court.

Dans la littérature, il y a beaucoup d'algorithmes qui permettent de chercher le chemin le plus court entre deux noeuds d'un graphe. Les plus connus sont certainement l'algorithme de Dijkstra (cf. [DIJ59]) et l'algorithme de Bellman-Ford (cf. [BEL57] et [FOR62]). Mais ces deux algorithmes ne trouvent le chemin le plus court qu'entre un noeud et tous les autres noeuds du réseau. Or, dans notre problème, il faut connaître le chemin de chaque noeud vers chaque autre noeud du réseau pour pouvoir déterminer la matrice de routage. Il faudrait alors effectuer l'algorithme de routage pour chaque noeud du réseau. De plus, il faudrait appliquer le tout après chaque transition afin de pouvoir calculer le coût total du réseau. Le temps de calcul va alors croître de plus en plus lorsque le nombre de noeuds augmente. Il faut donc trouver un autre algorithme de routage.

Il y a un algorithme qui calcule très rapidement la distance minimale entre tous les noeuds d'un réseau sans pour autant donner le chemin : l'algorithme de Floyd (cf. [FLO64]). Or, dans notre problème, on n'a pas besoin de connaître tout le chemin vers une destination; il suffit de connaître le premier noeud sur ce chemin. Tous les paquets vers cette destination seront alors envoyés vers le noeud voisin ; celui-ci s'occupera d'émettre les paquets vers le noeud suivant, etc.

Algorithme de Floyd

L'algorithme peut être défini de la manière suivante :

Soit N le nombre de noeuds dans le réseau.

d_{ij} la distance entre le noeud i et le noeud j s'ils sont reliés avec $d_{ii} = 0$ et $d_{ij} = \infty$ si les noeuds ne sont pas directement reliés

$D_{ij}^{(n)}$ la distance du chemin le plus court entre le noeud i et le noeud j avec la contrainte que seuls les noeuds 1, 2, ..., n peuvent être des noeuds intermédiaires sur ce chemin

L'algorithme de Floyd est alors constitué des étapes suivantes :

1. *Initialisation :*

$$D_{ij}^{(0)} = d_{ij} \quad \text{pour tout i,j tel que } i \neq j \qquad (4.1.)$$

2. *Itération :*

Pour tout n = 0, 1, 2, ..., N-1

$$D_{ij}^{(n+1)} = \min [D_{ij}^{(n)}, D_{i(n+1)}^{(n)} + D_{(n+1)j}^{(n)}] \quad \text{pour tout i,j tel que } i \neq j \qquad (4.2.)$$

Après l'initialisation, l'élément $D_{ij}^{(0)}$ est égal à la distance entre le noeud i et le noeud j s'il y a une ligne entre ces deux noeuds. Il prend la valeur infinie s'il n'y a pas de ligne entre les deux noeuds et la valeur 0 si i = j.

Après la première étape de l'itération, $D_{ij}^{(1)}$ est la distance entre le noeud i et le noeud j en passant par le noeud 1 si les noeuds i et j ne sont pas directement reliés. S'ils sont reliés par une ligne, $D_{ij}^{(1)}$ prend la valeur de la distance entre les deux noeuds.

Après l'étape deux, $D_{ij}^{(2)}$ représente le minimum de distance entre le noeud i et le noeud j en passant par le noeud 1 ou le noeud 2 ou par les deux si i et j ne sont pas reliés directement. Sinon c'est la longueur de la ligne directe i-j.

On répète ces étapes jusqu'à ce qu'on aboutisse au $n^{ième}$ calcul. Dans ce cas, on a calculé la distance minimale entre deux noeuds en pouvant passer par tous les noeuds du réseau. Ces calculs sont faits pour toutes les paires de noeuds dans le réseau.

Lorsqu'on veut connaître le noeud voisin sur le chemin dont la distance est minimale, il suffit de le mémoriser à chaque étape d'itération. Définissons donc :

$NV_{ij}^{(0)}$ est le noeud voisin vers lequel le noeud i doit envoyer les paquets pour que ceux-ci arrivent au noeud de destination j avec la contrainte que seuls les noeuds 1, 2, ..., n peuvent être des noeuds intermédiaires sur le chemin vers j.

Pour mémoriser ce noeud, il suffit alors d'ajouter les deux expressions ci-dessous :

- A l'initialisation, on ajoute l'égalité suivante à la formule (4.1.) :

$$NV_{ij}^{(0)} = \begin{cases} j & \text{si le noeud i est directement connecté au noeud j} \\ 0 & \text{si } i = j \\ -1 & \text{sinon} \end{cases} \qquad (4.3.)$$

  On mémorise donc le noeud voisin s'il y a une ligne entre les deux noeuds.

- A chaque itération, on complète la formule (4.2.) par :

$$NV_{ij}^{(n+1)} = \begin{cases} NV_{ij}^{(n)} & \text{si } D_{ij}^{(n+1)} = D_{ij}^{(n)} \\ NV_{i(n+1)}^{(n)} & \text{sinon} \end{cases} \qquad (4.4.)$$

  On garde le même noeud si la distance n'a pas changé. Si la distance diminue lorsqu'on passe par le noeud n+1, on prend le noeud voisin du chemin vers le noeud n+1.

Chapitre 5

Implémentation et évaluation de l'algorithme du recuit simulé

5.1. Implémentation et tests

Paramètres du recuit simulé

Nous allons expliquer dans cette section la manière dont on a cherché les paramètres du recuit simulé lors de l'application au problème de conception de réseaux distribués. On a vu que les paramètres de l'algorithme ont une signification physique. Malheureusement, ils n'en ont pas dans le cas de la conception de réseaux, ce qui rend la compréhension de leur fonction plus difficile. Par contre, on indiquera l'influence des paramètres sur le déroulement de l'algorithme. Il n'y a pas de méthode universelle pour trouver les valeurs des paramètres du recuit simulé. Pour chaque type d'application, on doit les trouver par des tests ou des expériences sur des exemples. Il faut alors généraliser les valeurs trouvées en établissant des formules qui reflètent l'évolution des paramètres en toute généralité. Il s'agit des paramètres suivants : le paramètre de contrôle c (la température), la loi de décroissance α (facteur décroissant), la longueur de la chaîne de Markov L_{Markov} (nombre d'itérations à l'intérieur d'un palier de température).

Ces paramètres influencent directement le nombre d'itérations (le nombre de fois qu'on applique des transformations) et l'acceptation d'une solution. Ainsi, ils déterminent indirectement la vitesse de convergence de l'algorithme et la qualité de la solution finale.

Il est clair que le nombre d'itérations doit changer lorsque le nombre d'hôtes et de noeuds augmente afin de pouvoir se balader dans tout l'espace de solutions à l'aide des transitions. Nous allons d'abord expliquer l'influence du nombre noeuds et hôtes sur l'algorithme et puis montrer comment trouver les paramètres propres à l'algorithme.

Le nombre d'hôtes et de noeuds

Comme on exige une connectivité d'ordre deux dans le cadre de ce mémoire, il faut aux moins trois noeuds pour pouvoir appliquer l'algorithme à une topologie de réseau. Tout problème de conception d'un ou de deux noeuds est d'ailleurs relativement facile et la solution peut être trouvée par une recherche exhaustive.

De plus, si le nombre d'hôtes augmente, on exigera que le nombre de noeuds augmente aussi. Pour éviter d'avoir des réseaux avec un trop grand nombre de noeuds lorsqu'on a beaucoup d'hôtes, on fait croître le nombre de noeuds plus lentement que le nombre d'hôtes. Cette mesure empêche d'avoir une convergence trop lente dans le cas des réseaux de grande taille. Ceci a comme conséquence que le nombre d'hôtes par noeud diminue lorsque le nombre d'hôtes augmente. Le *Tableau 5.1* montre la relation entre le nombre d'hôtes H et le nombre de noeuds (la limite supérieure NT_{max} et la limite inférieure N_{min}). Si on a par exemple 80 hôtes, le nombre maximal de noeuds NT_{max} sera compris entre 20 et 25, on aura au maximum 23 noeuds. Ceci donne une moyenne de 3,5 hôtes par noeuds. Par contre, si on prend par exemple 400 hôtes, le

nombre maximal de noeuds sera 47, ce qui correspond à une moyenne de 8,5 hôtes par noeuds.

H (nbr d'hôtes)	NT_{max} (nbr max de noeuds)	N_{min} (nbr min de noeuds)
≤ 60	$3 \rightarrow 20$	3
$61 \rightarrow 100$	$21 \rightarrow 25$	$4 \rightarrow 5$
$101 \rightarrow 150$	$26 \rightarrow 30$	$5 \rightarrow 6$
$151 \rightarrow 210$	$31 \rightarrow 35$	$6 \rightarrow 7$
$211 \rightarrow 280$	$36 \rightarrow 40$	$7 \rightarrow 8$
$281 \rightarrow 360$	$41 \rightarrow 45$	$8 \rightarrow 9$
$361 \rightarrow 450$	$46 \rightarrow 50$	$9 \rightarrow 10$
> 450	> 50	> 10

Tableau 5.1 : Quantité de noeuds par rapport au nombre d'hôtes

Comme on ne peut pas placer plus de noeuds qu'il y a des positions, le nombre maximal de noeuds que l'on acceptera dans l'algorithme est déterminé le nombre de positions potentielles. Voici les expressions permettant de calculer les bornes du nombre de noeuds :

$$N_{max} = \max (3 , \min (P , NT_{max}))$$
$$N_{min} = \max (3 , [N_{max} / 5]) \qquad (5.1.)$$

Comme déjà dit, lorsque le nombre de noeuds croît, il faut que le nombre d'itérations augmente aussi. Comme le nombre de combinaisons de noeuds détermine le nombre de solutions, il suffit de le calculer pour avoir une idée de l'ordre de grandeur du volume de solutions. Une fois qu'on connaît le volume de solutions, on peut fixer le nombre d'itérations pour parcourir l'espace des solutions. Comme les paramètres de l'algorithme déterminent le nombre d'itérations, on va essayer de retrouver les valeurs de ceux-ci à partir du nombre d'itérations.
En se basant sur les deux bornes ci-dessus, on peut calculer le nombre de combinaisons possibles de placer entre N_{min} et N_{max} noeuds parmi les P positions potentielles. Le résultat de ce calcul est très intéressant pour avoir une idée du volume de transitions possibles dans l'algorithme du recuit simulé. Si on connaît le nombre de transitions, on peut déduire la valeur des paramètres pour trouver une solution satisfaisante.

Calculons d'abord le nombre de solutions dans l'espace des solutions. L'analyse combinatoire nous enseigne que le nombre de combinaisons possibles lorsqu'on choisit K noeuds parmi N positions (sélection de K éléments parmi N sans remise) est donné par l'expression :

$$C_N^K = \frac{N!}{K!(N-K)!} \qquad (5.2.)$$

Or, dans notre cas, on peut avoir entre N_{min} et N_{max} noeuds dans le réseau. Pour avoir le nombre total de combinaisons de tous les noeuds, il faut faire la somme des combinaison. Ceci s'exprime de la manière suivante :

$$C_{total} = \sum_{K=N_{min}}^{N_{max}} C_P^K \qquad (5.3.)$$

Or, la forme simple du binôme de Newton :

$$\sum_{K=0}^{P} C_P^K = 2^P \qquad (5.4.)$$

permet de simplifier l'expression (5.3.).

Ainsi, lorsque le nombre maximal de noeuds N_{max} est égal au nombre de positions potentielles P et si N_{min} est égal au minimum (donc 3), il y a moins que 2^P combinaisons possibles; il y en a exactement :

$$\begin{aligned} \sum_{K=3}^{N_{max}} C_P^K &= 2^P - (C_P^2 + C_P^1 + C_P^0) \\ &= 2^P - (P\cdot(P+1)/2 + 1) \end{aligned} \qquad (5.5.)$$

Calculons - à titre d'exemple - le nombre de combinaisons sur un réseau de 40 hôtes et de 10 positions potentielles. En remplaçant le nombre d'hôtes H et le nombre de positions potentielles P par les valeurs correspondantes dans les deux équations de (5.1.), on trouve N_{max} et N_{min} :

$$\begin{aligned} N_{max} &= \max(3, \min(10, [40/3])) \\ &= \max(3, \min(10, 13)) = 10 \\ N_{min} &= \max(3, 2) = 3 \end{aligned}$$

L'algorithme peut donc placer entre 3 et 10 noeuds parmi les 10 positions potentielles existantes. Le nombre de combinaisons possibles pour placer les noeuds est le suivant :

$$C_{total} = 2^{10} - (10 * (10+1))/2 - 1 = 1024 - 55 - 1 = 968$$

Nombre d'itérations

Comme déjà dit, le nombre de combinaisons de noeuds (donc la taille de l'espace des solutions) influence le nombre total d'itérations. Comme l'algorithme du recuit simulé itère par paliers de temps (paliers de paramètre de contrôle) en diminuant le paramètre c par un facteur α, le nombre total d'itérations dépend de c. De plus, à l'intérieur d'un palier de température, on effectue un certain nombre de transitions (itérations) qui est déterminé par la longueur des chaînes de Markov de solution réalisable. Pour y garantir un équilibre, la chaîne de Markov doit être infinie. Comme nous l'avons vu au chapitre précédent, il suffit d'une chaîne finie, mais assez longue, pour obtenir un quasi-équilibre et d'avoir des résultats satisfaisants.

Le paramètre de contrôle c et la longueur des chaînes de Markov déterminent donc le nombre total d'itérations. Or, la taille de l'espace des solutions croît de façon exponentielle avec le ombre qu'on peut placer dans le réseau (c.f. 5.5). Pour éviter une croissance exponentielle du nombre d'itérations, nous allons effectuer un nombre raisonnable d'itérations afin de limiter le temps de convergence.

Cherchons d'abord une expression permettant de préciser la longueur L_{Markov} de la chaîne de Markov. Les nombreux tests qu'on a fait pour trouver des bonnes valeurs des paramètres nous ont également permis de trouver les valeurs pour la longueur des chaînes de Markov pour certains exemple et d'estimer l'évolution des valeurs de L_{Markov}. Le *Tableau 5.2* montre le résultat de ces estimations dans les deux premières colonnes :

N_{max}	L_{Markov} estimée	L_{Markov} effective
5	20	15
10	100	90
15	400	405
20	1500	1620
25	6000	6075

Tableau 5.2 : Longueur des chaînes de Markov

On a alors tenté de généraliser ces valeurs dans une formule. En exigeant un minimum de 15 itérations, les valeurs de la deuxième colonne du tableau ci-dessus se résument dans l'expression :

$$L_{Markov} = \max (15 , N_{max} * 3^{(N_{max}/5)}) \qquad (5.6.)$$

Les valeurs trouvées par cette formule sont reprises dans la troisième colonne du *Tableau 5.2.*

Revenons à notre exemple de 40 hôtes et 10 positions potentielles, pour lequel $N_{max} = 10$ et $N_{min} = 3$. Conformément à la formule (5.6.), la longueur de la chaîne de Markov de cet exemple vaut :

$$L_{Markov} = \max (15 , 10 * 3^{(10/5)}) = 10 * 3^2 = 90$$

Après avoir trouvé la longueur de la chaîne de Markov (donc le nombre d'itérations à l'intérieur d'un palier), on doit encore trouver le facteur de décroissance α pour déterminer le nombre total d'itérations. Comme déjà dit, ce paramètre détermine le nombre de chaînes de Markov (le nombre de paliers) à générer. Au chapitre précédent, nous avons vu que différents auteurs ont montré qu'il faut prendre des valeurs de α entre 0,85 et 0,99.

On doit de nouveau tenir compte du fait que le nombre de combinaisons augmente de façon exponentielle, lorsque le nombre de noeuds augmente. Il faut donc générer beaucoup de chaînes de Markov pour trouver une solution satisfaisante. Autrement dit, dans la plage du paramètre de contrôle c entre sa valeur initiale et finale (entre c_{max} et c_{min}), on doit avoir plus de paliers si le nombre de noeuds augmente. Il y a deux manières de le faire : augmenter la différence entre c_{max} et c_{min} ou augmenter la valeur du facteur décroissant α. Mais, comme le paramètre de contrôle c influence également l'acceptation ou le rejet d'une solution dans le critère de Metropolis, il est plus prudent de seulement modifier α. Ici, les tests nous ont également permis d'estimer les valeurs de α en fonction du nombre de noeuds (*Tableau 5.3*).

N_{max}	α estimé	α effectif
3	0,85	0,85
5	0,87	0,86
10	0,92	0,90
20	0,95	0,95
30	0,97	0,975
40	0,98	0,985
>50	0,99	0,994

Tableau 5.3 : Estimation du facteur de décroissance

L'estimation du comportement du facteur α est résumé dans la formule suivante dont le résultat est indiqué dans la troisième colonne du tableau ci-dessus :

$$\alpha = \max (0{,}85 ; \min(0{,}999 ,1 - 0{,}2/2^{(N_{max}/10)})) \qquad (5.7.)$$

Cherchons le facteur de décroissance α pour notre exemple. En tenant compte de la formule (5.7.), on trouve la valeur suivante pour α :

$$\alpha = \max (0{,}85 ; \min(0{,}999 ,1 - 0{,}2/2^{1})) = \max(0{,}85 ; 0{,}90) = 0{,}90$$

Le paramètre de contrôle

Maintenant, il faut encore trouver les limites supérieures et inférieures du paramètre de contrôle c (c_{max} et c_{min}). Comme déjà dit, la valeur du paramètre c intervient dans le critère de Metropolis et influence donc l'acceptation des solutions.

La valeur maximale c_{max} doit être telle que le critère de Metropolis accepte toute transition, qui ne diminue pas le coût, avec une probabilité de 0,5. En tenant compte de la formule (4.2.), elle s'écrit

$$\exp(-\Delta f/c_{max}) = 0{,}5$$

ou encore

$$-\Delta f/c_{max} = -0{,}7 \qquad (5.8.)$$

Or, les tests ont montrés que le coût moyen d'une transition est de l'ordre de grandeur du coût d'une ligne. Malheureusement, le coût optimal du réseau n'est pas connu; il faut donc l'estimer. Comme il est assez difficile de faire une estimation du coût sans informations supplémentaires, le programme va calculer la solution initiale. Pour avoir une idée du coût total du réseau, on calcule le coût de la solution initiale. En divisant celui-ci par le nombre de lignes dans la solution initiale, on aura le coût moyen par ligne et donc la différence de coût :

$\Delta f = f_{init} / NL_{init}$ où f_{init} = coût de la solution initiale
NL_{init} = nombre de ligne de la solution initiale

En tenant compte de cette expression et de (5.8.), la valeur maximale du paramètre de contrôle se calcule par l'expression :

$$c_{max} = f_{init} / (NL_{init} * 0,7) \qquad (5.9.)$$

La valeur minimale c_{min} doit être telle que le critère de Metropolis accepte seulement des solutions qui diminuent le coût f ou des solutions qui augmentent le coût avec une probabilité de :

$$\exp(-\Delta f/c_{min})$$

Les tests ont montré qu'il faut refuser une différence de coût Δf de ±1 % du coût global avec une probabilité très élevée (99 %). Autrement dit, la probabilité d'accepter une solution plus cher que 1 % du coût total ne peut pas 0,01 (1%). Plus précisément, il faut que

$$\exp(-\Delta f/c_{min}) = 0,01$$

ou encore

$$-\Delta f/c_{min} = -4,5 \qquad (5.10.)$$

En se basant de nouveau sur le coût de la solution initiale f_{init}, la différence de coût est donnée par

$$\Delta f = f_{init} / 100$$

A l'aide des deux dernières formules, la valeur du paramètre de contrôle c_{min} peut être trouvée par la relation suivante :

$$c_{min} = f_{init} / 450 \qquad (5.11.)$$

Par exemple pour un réseau dont le coût initial f_{init} vaut 2 millions de FB et dont la solution initiale est constituée de 20 lignes (NL_{init}), c_{max} est égal à 142.857 et c_{min} vaut 4444. Remarquons que les coûts sont exprimés en milliers de FB dans le programme; le paramètres de contrôle est également exprimé en milliers de FB. Le programme va donc afficher les valeurs suivantes : $c_{max} = 143$ et $c_{min} = 4$.

Remarquons que les valeurs de tous ces paramètres donnent simplement une idée de leur ordre de grandeur. Comme chaque réseau à ses propres caractéristiques, il est impossible de donner des valeurs optimales pour tout type de réseau. Le lecteur intéressé peut modifié les paramètres dans le programme joint à ce mémoire pour voir les effets sur la solution de l'algorithme de conception de réseaux distribués suite à un changement de paramètres.

Tests effectués

Passons maintenant à l'explication d la manière dont on a implémenté l'algorithme du recuit simulé.

Avant d'effectuer des tests, il fallait accomplir différentes tâches. La formalisation de l'organigramme sous forme mathématique est une tâche importante et cruciale. Les autres tâches sont la vérification des spécifications et finalement la traduction de ces expressions en commandes du langage de programmation. Après avoir vérifié que les commandes correspondent bien aux spécifications, on est rentré dans la phase des tests où on applique algorithme à des exemples.

Notons que chaque "sous-algorithme" a d'abord été testé seul sur un exemple approprié pour être finalement mis ensemble. Remarquons également que chacune de ces étapes a été effectuée pour les algorithmes (algorithme du recuit simulé, algorithme dual de l'arbre minimal, algorithme d'accès local et algorithme de routage) ainsi que les autres calculs et modules dans le programme (p.ex. vérification de la connectivité, calcul des capacités, calcul des coût, calcul des performances, calcul de la distance entre deux points, gestion de listes et tableaux, etc. ...). Lorsque tout a été implémenté et testé sur son bon fonctionnement, on a pu commencer avec les tests pour trouver les paramètres de l'algorithme du recuit simulé.

Toutes les formules et expressions ci-dessus permettant de déterminer les paramètres de l'algorithme sont basées sur de nombreux tests. A l'aide de ces tests, nous avons pu voir et prédire le comportement de l'algorithme en modifiant les paramètres et nous avons finalement pu déduire les formules décrites ci-dessus. Nous allons maintenant présenter l'approche qu'on a suivie durant les tests.

- Etant donnée la structure de l'algorithme, il se peut que l'algorithme tombe par hasard très tôt sur la solution optimale (optimum global) et quitte cette solution à cause du critère de Metropolis. L'algorithme peut retrouver cette solution à la fin de l'exécution de l'algorithme, mais il peut aussi bien trouver une solution assez éloignée de l'optimum. Un moyen d'éviter ce dernier cas et de contrôler la convergence vers l'optimum est de comparer chaque solution à la solution minimal obtenue jusque là et considérer celle-ci comme nouvelle solution minimale si son coût est inférieur. Ainsi, on peut également vérifier si la solution finale est bien l'optimum que l'algorithme peut trouver.

- L'étape suivante consiste à générer un très grand nombre de transitions et d'accepter beaucoup de solutions (même si la différence des coûts est très élevée) espérant ainsi de tomber sur la solution optimale. Pour cela il faut associer de grandes valeurs aux paramètres de contrôle c_{max} et prendre de longues chaînes de Markov. Etant donné ces valeurs, l'exécution de l'algorithme prend beaucoup de temps. Comme on génère énormément de solutions, la probabilité de trouver la solution minimale est très élevée. Remarquons que lors de ces test, nous avons également cherché la solution d'un petit exemple par une recherche exhaustive (à l'aide d'un tableur) pour savoir si la solution trouvée par l'algorithme était l'optimum global ou seulement un optimum local. Pour les réseaux testés, la solution trouvée par l'algorithme était bien un optimum global.

- Une fois qu'on connaît la solution optimale, il suffit de réduire la valeur des paramètres afin de diminuer le nombre d'itérations et par conséquent le temps de calcul. Nous avons pu constater deux comportements distincts lorsqu'on diminue ces paramètre. Premièrement, si la valeur minimale du paramètre de contrôle c_{min} est trop élevée, l'algorithme accepte vers la fin trop de solutions qui ne diminuent pas le coût. Ceci entraîne que la solution finale peut être assez éloignée de la solution optimale. Deuxièmement, si la longueur de la chaîne de Markov est trop courte, l'algorithme génère trop peu de transitions pour pouvoir couvrir la majorité de l'espace des solutions. En conséquence, les solutions optimales trouvées par l'algorithme sont souvent des solutions distinctes; ce qui veut dire qu'on n'a atteint pas l'équilibre à l'intérieur des paliers.

5.2. Application concrète

Exemple

Afin de mieux comprendre l'algorithme du recuit simulé appliqué aux réseaux distribués, nous allons l'illustrer sur un exemple. On montrera le fonctionnement général de l'algorithme sans entrer dans les détails de chaque étape. Prenons comme exemple un réseau de cinq hôtes et de quatre positions potentielles (*Figure 5.1*).

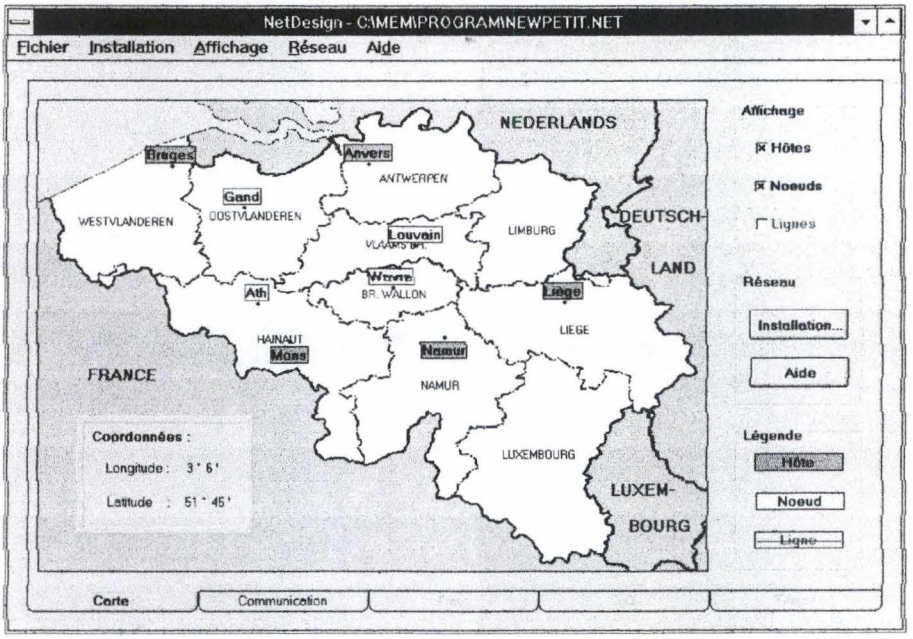

Figure 5.1 : Réseau de 5 hôtes et 4 positions

Supposons que la matrice de communication soit donnée par la matrice suivante (*Figure 5.2*) :

	Anvers	Bruges	Liège	Mons	Namur
Anvers	0	50	50	20	30
Bruges	60	0	30	20	20
Liège	40	40	0	30	50
Mons	20	30	40	0	60
Namur	20	20	60	50	0

Figure 5.2 : Matrice de communication

Comme il y a cinq hôtes, déduisons de la formule (5.1.) le nombre maximal de noeuds $N_{max} = \max(3,5/3) = 3$. Le nombre minimal vaut également 3. L'algorithme peut donc choisir exactement trois noeuds parmi les quatre positions. Il y a en tout

$$C_4^3 = \frac{4!}{3!1!} = 4$$

combinaisons possibles. On peut les voir sur la *Figure 5.3*.

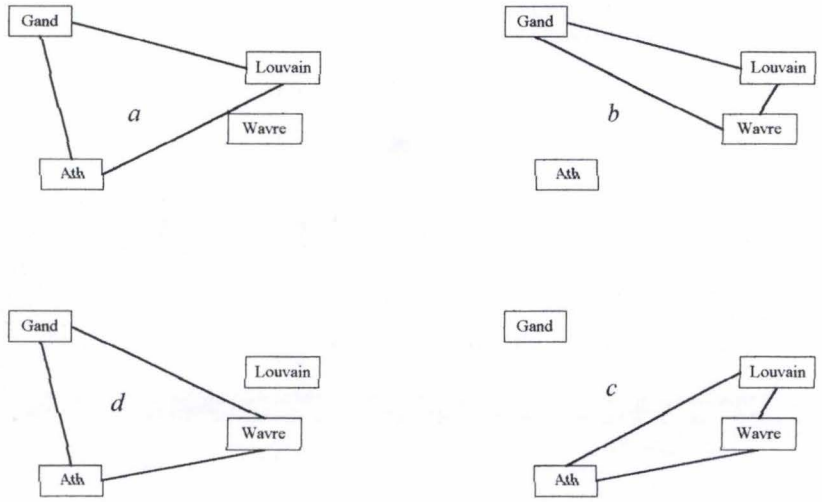

Figure 5.3 : Les quatre combinaisons possibles

Le programme calculera alors les paramètres de l'algorithme du recuit simulé à l'aide des formules décrites dans la section précédente. La *Figure 5.4* montre ces paramètres.

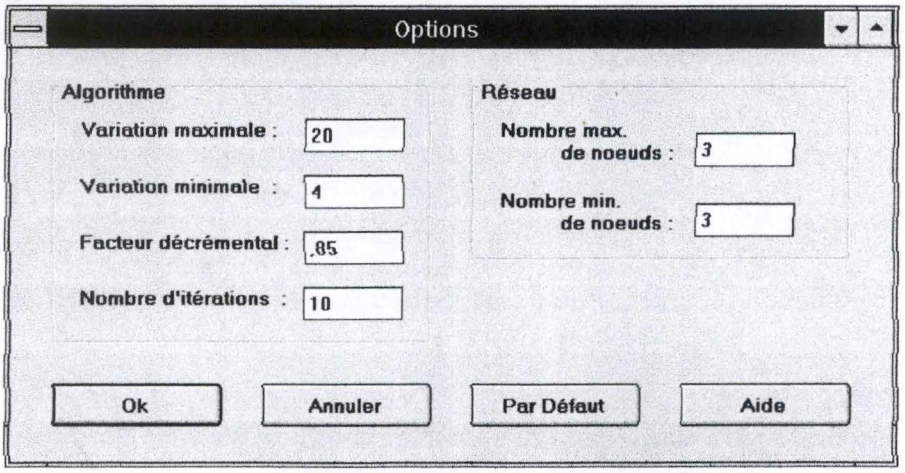

Figure 5.4 : Paramètres par défaut

L'algorithme du recuit simulé va d'abord générer une solution initiale; par exemple la solution de la *Figure 5.3c*. Il calcule alors le trafic moyen et les capacités à l'aide de la matrice de communication et de la matrice de routage. Une fois que les capacités sont connues, on doit déterminer le coût des lignes en tenant compte des distances entre les machines et finalement le coût total du réseau. Ces distances se déduisent de la longitude et de la latitude des hôtes et des positions potentielles. Le coût total de la solution initiale de notre exemple vaut 522.457 FB.

A la première itération, le paramètre de contrôle c prend sa valeur maximale c_{max}. On répète alors les étapes suivantes tant que le paramètre c est plus grand que sa valeur minimale c_{min} :

Pour chaque valeur du paramètre de contrôle, on applique 10 (la longueur de la chaîne de Markov) transitions. Une transition est composée de plusieurs étapes. On décide d'abord par hasard d'un déplacement, d'un ajout ou d'un retrait d'un noeud. L'algorithme du réseau d'accès local relie alors les hôtes aux noeuds les plus proches. Ensuit, l'algorithme dual de l'arbre minimal place les lignes entre noeuds. Si toutes les lignes sont placées, l'algorithme de routage établit la matrice de routage en fonction des distances entre noeuds.

Supposons que la première transition mène à la configuration *b* de la *Figure 5.3*. Il faut alors calculer la capacité des lignes et leur coût. Dans le cas *b*, celui-ci vaut 504.224 FB. L'algorithme calcule alors la différence de coût et accepte la nouvelle solution si le coût diminue, ce qui est le cas ici. Dans le cas contraire, il applique le critère de Metropolis, lequel va déterminer l'acceptation à l'aide de la formule (4.2). Une fois que la solution est acceptée, on continue sur cette solution. Si elle est rejetée, on revient sur la configuration précédente. On génère alors la transition suivante.

Remarquons encore que la solution optimale de l'exemple est représentée la *Figure 5.3c* avec un coût total de 504.224 FB.

Dans l'exemple simple ci-dessus, l'algorithme du recuit simulé a pu diminuer le coût total du réseau de 522.457 FB à 504.224 FB ce qui correspond à une diminution du coût de 3,5 %.

Pour montrer des diminutions de coût plus importantes, nous avons appliqué l'algorithme du recuit simulé à un autre exemple. Cet exemple est composé de 17 hôtes et 7 positions potentielles (il est également sur la disquette jointe à ce mémoire). Notons que le coût d'une solution générée aléatoirement parmi les solutions initiales s'élève à 2.624.101 FB (réseau de 6 noeuds). L'algorithme trouve une solution optimale d'un coût de 2.231.197 FB. Ce qui correspond à une diminution de 15 %.

Environnement

Le programme qui accompagne ce mémoire est écrit en Visual Basic (version 3.0). Le langage de programmation Visual Basic a été choisi à cause des facilités pour représenter la solution au problème de conception sous forme graphique. Visual Basic permet en effet d'introduire facilement des objets prédéfinis comme par exemple des boutons, images, boîtes à cocher, etc.

Le programme a été écrit sur un PC Pentium 133 MHz avec 32 MB de mémoire RAM et un écran 15 pouces. Le programme tourne et a été testé sous Windows 95. Remarquons qu'il faut une résolution de 1024×728 points (pixels) pouces pour voir correctement la carte de la Belgique avec la position exacte des hôtes et noeuds. Les tests ont été effectués dans le même environnement. En ce qui concerne l'exemple des 17 hôtes et 7 positions potentielles, il faut entre 60 et 70 secondes.

5.3. Discussion des résultats

Les chapitres précédents ont montré que le problème de la conception de réseaux et plus particulièrement de réseaux distribués est un problème assez complexe. Il faut tenir compte de beaucoup d'éléments qui doivent être intégrés dans l'algorithme : le nombre et l'emplacement des noeuds, le placement des lignes entre noeuds en garantissant la connectivité d'ordre deux, la connexion des hôtes à leur noeud voisin, le calcul du coût d'une ligne (ayant une certaine capacité) suivant le système de tarification du fournisseur; établissement de la matrice de routage, etc.

Suite à cette complexité, il est clair qu'il n'existe pas d'algorithme pouvant prétendre de trouver avec certitude la solution optimale au problème posé endéans un temps de convergence raisonnable. Il en est de même pour l'approche présentée dans le cadre de ce mémoire. L'algorithme du recuit simulé trouve de résultats acceptables après un temps de calcul relativement court. Remarquons qu'il y a probablement encore d'autres approches et d'autres algorithmes qui permettent de trouver également de bonnes ou meilleures solutions.

Outre la complexité du problème de conception de réseaux, de nombreux tests ont été nécessaires pour, d'une part, assurer le fonctionnement correct des différents algorithmes (recuit simulé, dual de l'arbre minimal, algorithme des réseaux d'accès local, vérification de la connectivité d'ordre deux, algorithme de routage, ...) et , d'autre part, pour déterminer les valeurs des paramètres (paramètre de contrôle c, facteur décroissant α, longueur des chaînes de Markov) permettant de trouver la meilleure solution que l'algorithme peut trouver dans des temps de convergence raisonnables.

Examinons maintenant le comportement de l'algorithme du recuit simulé. Comme nous l'avons déjà vu au chapitre quatre, le recuit simulé a deux avantages

majeurs : D'une part, il parcourt l'espace des solutions de manière telle qu'il peut encore sortir d'un minimum local dans le but de se diriger vers l'optimum global (*Figure 4.3*). D'autre part, cet algorithme peut relativement facile calculer la différence de coût entre deux solutions voisines dans beaucoup d'applications.

Malheureusement, dans le cas de la conception des réseaux, le changement effectué sur un seul élément du réseau modifie le comportement complet de ce dernier. Plus précisément, l'ajout, le retrait ou le déplacement d'une ligne ou d'un noeud change complètement les flux de communication : les paquets peuvent suivre des chemins totalement différents. Ceci influence la capacité des lignes et par conséquent le coût de beaucoup de lignes est modifié sans pouvoir le prédire d'avance. Lorsqu'on calcule donc la différence de coût entre deux solutions voisines, il faut tenir compte des modifications dans tout le réseau; ce qui veut dire qu'on doit recalculer les capacités et les coûts de toutes les lignes du réseau. Malgré que cette propriété du recuit simulé ne soit pas applicable dans notre cas, il nous semble que le fait de pouvoir sortir de minima locaux est un phénomène très intéressant qu'on trouve dans peu d'autres algorithmes.

Il reste encore à mentionner un phénomène intéressant qu'on a constaté. Lors des tests effectués après l'implémentation de l'algorithme, on a constaté que l'algorithme du recuit simulé implémenté tente à sélectionner le moins de noeuds possibles. Plus précisément, lorsqu'on a une topologie pour laquelle on propose par exemple 10 positions, parmi lesquelles l'algorithme peut choisir entre 4 à 8 noeuds, l'algorithme va construire un réseau de quatre noeuds et relier les hôtes à son noeud voisin.

L'explication possible de ce phénomène peut être la suivante :

- Soit, la capacité des lignes entre hôtes et noeuds est toujours la même, indépendamment du noeud auquel l'hôte est relié. Ce qui veut dire que le coût des lignes hôte-noeud ne dépend que de la distance entre l'hôte et son noeud voisin. Or, comme les tarifs du fournisseur sont connus, les coûts fixes sont assez élevés par rapport aux coûts liés à la distance. Le coût global d'une ligne hôte-noeud augmente donc relativement peu lorsqu'on connecte un hôte à un noeud plus éloigné.
- Soit, s'il y a moins de noeuds, il y a d'une part peu de lignes entre noeuds et d'autre part plusieurs hôtes sont reliés à un même noeud. Or, comme le trafic entre hôtes directement liés au même noeud traverse seulement ce noeud, le trafic ne passe pas sur les lignes entre noeuds. S'il y a donc beaucoup d'hôtes connectés à un même noeud, il y a peu de trafic sur les lignes entre noeuds et par conséquent leur capacité et leur coût est faible. En résumé, on peut dire que le coût global des lignes entre noeuds décroît fortement si le nombre de noeuds diminuer et que le coût global des lignes entre hôtes et noeuds croît légèrement dans le même cas. Le coût global du réseau est finalement plus petit s'il y a peu de noeuds.
- Soit l'interaction des deux explications décrites ci-dessus.

Chapitre 6

Conclusion

Dans une période où l'homme est de plus en plus jugé en termes de performance, la circulation rapide et efficace de l'information est un élément-clé dans les entreprises d'aujourd'hui : nous nous trouvons actuellement dans l'ère de l'information et de la communication. Et qui dit communication, dit communication à distance : la télécommunication.

De plus en plus d'entreprises instaurent des réseaux de communication pour faciliter le flux de communication entre personnes - ou machines - distantes. L'installation et surtout l'utilisation de ces réseaux à grande distance représentent des frais énormes. Tout outil permettant de limiter ces dépenses est le bienvenu. Ce travail se voit justement comme un tel outil.

Le but de ce mémoire consiste à contribuer à la conception de réseaux de télécommunication. Comme il y a beaucoup de types de réseaux différents, nous nous sommes limités à un type de réseau bien particulier : les réseaux distribués privés a commutation par paquets (de taille fixe) empruntant des lignes louées.

Ce travail est composé de deux parties principales : d'une part le calcul des capacités et des coûts de lignes et d'autre part le problème de la conception de réseaux.

On a défini dans la première partie les composantes du réseau pour passer ensuite au calcul du trafic moyen par ligne à partir de la matrice de communication et de la matrice de routage. En tenant compte du critère de performance de 60 % (taux d'occupation), nous avons établi les formules pour déterminer la capacité réalisable des lignes. La grille de tarification du fournisseur des lignes permet alors de chiffrer le coût par ligne et finalement le coût total du réseau. Nous avons calculé, à titre d'information, les temps de réponse à l'aide des formules de la théorie des files d'attente.

Le point central de la deuxième partie est la conception de réseaux. Nous avons d'abord classifié les réseaux en différentes catégories principales : réseaux centralisés et réseaux distribués. Ces derniers sont composés des réseaux d'accès local et du réseau de transit. Cette classification a facilité la compréhension des algorithmes de conception trouvés dans la littérature spécialisée. Nous avons alors expliqué l'approche suivie dans ce mémoire avant d'entamer l'étude des différents algorithmes. Les algorithmes suivants ont été présentés, analysés et finalement comparés :

- pour les réseaux centralisés : l'algorithme d'insertion et l'algorithme d'élimination,
- pour les réseaux d'accès local : l'algorithme de placement des noeuds et l'algorithme des réseaux d'accès local,
- pour les réseaux de transit : l'algorithme de Steiglitz, l'algorithme dual de l'arbre minimal et le "Cut Saturation algorithm",
- pour les réseaux distribués en général : l'algorithme du recuit simulé.

Une étude plus approfondie des aspects physiques en analytiques du recuit simulé précède les explications spécifiques au problème de la conception de réseaux distribués. Un exemple illustre la théorie. Lors la discussion des résultats, nous avons constaté que l'algorithme du recuit simulé tente à placer un minimum de noeuds.

Tous les concepts ainsi que quelques algorithmes décrits dans ce travail sont intégrés dans le programme du mémoire.

Remarquons qu'il a été assez difficile de déterminer les paramètres de l'algorithme du recuit simulé et de les formaliser. Cette difficulté montre qu'il faut une étude fondée du réseau, sur lequel on applique l'algorithme, pour trouver des solutions satisfaisantes.

Malgré qu'on a trouvé un algorithme qui permet de diminuer le coût de quelques réseaux distribués, il se pose la question si le calcul des coûts et l'utilisation des algorithmes d'optimisation présentés dans ce travail peuvent résoudre le problème de la conception de réseaux distribués de manière satisfaisante. Afin de s'approcher de ce but, on peut encore envisager des modifications pour améliorer les performances de l'algorithme.

Une modification envisageable serait de construire le réseau de transit non par l'algorithme dual de l'arbre minimal, mais par l'algorithme de Steiglitz ou un autre algorithme existant dans la littérature. Il serait alors intéressant de comparer les solutions trouvées par les deux approches.

On pourrait également appliquer l'algorithme du recuit simulé à un réseau existant et comparer la solution trouvée aux coûts réels du réseau. Ceci permettrait de savoir si l'algorithme est vraiment utilisable dans le monde des télécommunications. D'autres tests sur des réseau de grande taille seraient intéressant lorsqu'on connaît l'optimum absolu afin de situer la qualité des solutions trouvées par l'algorithme du recuit simulé.

Ceci ne sont que quelques suggestions pour continuer la recherche dans le domaine de la conception de réseaux.

En conclusion, on peut dire qu'il n'y a pas encore d'algorithme qui permettrait de résoudre tous les problèmes de conception. Aujourd'hui, les outils existants de la conception se limitent généralement à des réseaux bien déterminés et bien définis. L'importance croissante des réseaux de télécommunication nécessite des outils informatiques qui soutiennent la conception de tels réseaux.

On constate donc combien la conception de réseaux peut être un domaine de recherche émouvant et excitant...

Liste des figures et tableaux

Figures

Tableaux

Bibliographie

[AAR89] Aarts, E. and J. Korst
Simulated Annealing and Boltzmann Machines
John Wiley & Sons Ltd., Essex, 1989

[BAH72] Bahl, L.R. and D.T. Tang
Optimization of Concentrator Locations in Teleprocessing Networks
Proceedings Symposium on Computer-Communication Networks and Teletraffic, Polytechnical Institute of Brooklyn, New York, 1972, pp. 355-362

[BEL57] Bellman, R.E.
Dynamic Programming
Princeton University Press, Princeton, New Jersey, 1957

[COY95] Coydon, B.
Visual Basic 4
Eyrolles, Paris, 1995

[DIJ59] Dijkstra, E.
A Note on Two Problems in Connection with Graphs
Numerical Mathematics 1, octobre 1959, pp. 269-271

[FLO64] Floyd, R.W.
cf. Théorie des Graphes, cours donné aux FUNDP, Faculté des Sciences, Département de Mathématiques, Namur, 1994

[FOR62] Ford, L.R. and D.R. Fulkerson
Flows in Networks
Princeton University Press, Princeton, New Jersey, 1962

[GAU93] Gautschi, M.
Genetische Algorithmen für den Entwurf von Telekommunikationsnetzen
Institut für Informatik und angewandte Mathematik, Diplomarbeit, Universität Bern, 1993

[GER74] Gerla, M., H. Frank, W. Chou and J. Eckl
A cut saturation Algorithm for Topological Design of Packet Switched Communications networks
Proceedings of the National Telecommunications Conference, 1974, pp. 1074-1085

[GRE93] Greulich, A.
Übersicht über weitere Methoden der Netzwerkoptimierung
Institut für Informatik und angewandte Mathematik, Universität Bern, 1993

[GRE93a] Greulich, A.
Problembeschreibung zur Netzwerkoptimierung
Institut für Informatik und angewandte Mathematik, Universität Bern, 1993

[HUI95] Huitema, C.
Le routage dans l'Internet
Eyrolles, Paris, 1995

[KIR83] Kirkpatrick, S., C.D. Gelatt jr. and M.P. Vecchi
Optimization by simulated annealing
Science 220, 1983, pp. 671-680

[LAA87] van Laarhoven, P.J.M. and E.H.L. Aarts
Simulated Annealing: Theory and Applications
D. Reidel Publishing Company, 1987

[MAR72] Martin, J.
Systems Analysis for Data Transmission
Prentice-Hall, Englewood Cliffs, New Jersey, 1972

[MET53] Metropolis, J. et al.
Equation of State Calculations by fast Computing Machines
J. of Chem. Physics 21, 1953, pp. 1087-1092

[NIJ75] Nijenhuis, A. and H.S. Wilf
Combinatorial Algorithms
Academic Press, New York, 1975

[NUS87] Nussbaumer, H.
Téléinformatique II
Presses polytechniques romandes, Lausanne, 1987

[PRI57] Prim, R.C.
Shortest Connection Networks and some Generalisations
Bell Systems Technical Journal 36, novembre 1957, pp. 1389-1401

[SCH77] Schwartz, M.
Computer-Communications Network Design and Analysis
Prentice-Hall, Englewood Cliffs, New Jersey, 1977

[SHA90] Sharma, R.L.
Network Topology Optimization: the Art and Science of Network Design
Van Nostrand Reinhold, New York, 1990

[SIA88] Siarry, P. et G. Dreyfus
La méthode du recuit simulé: théorie et applications
Ecole Supérieure de Physique et de Chimie Industrielles de la Ville de Paris, I.D.S.E.T., Paris, 1988

[SRI95] Srivatsa, S.K. and P. Seshaiah
On the topological design of Computer Network
Computer Networks and ISDN Systems 27, 1995, pp. 567-569

[STA94] Stallings, W.
Data and Computer Communications
Prenctice Hall International, New Jersey, 1994

[STE69] Steiglitz, K., P. Weiner and D.J. Kleitman
The Design of Minimum-Cost Survivable Networks
IEEE Transactions on Circuit Theory, vol. CT-16, novembre 1969, pp. 455-460

[TUC80] Tucker, A.
Applied Combinatorics
John Wiley & Sons Ltd, Toronto, 1980

[WEB93] Weber, S.
Optimierung von Telekommunikationskosten in Unternehmen
Institut für Informatik und angewandte Mathematik, Inauguraldissertation, Universität Bern, 1993

[WOO73] Woo, L.S. and D.T. Tang
Optimization of Teleprocessing Networks with Concentrators
Proceedings IEEE National Telecommunications Conference, Atlanta, 1973, pp. 37C-1 à 37C-5